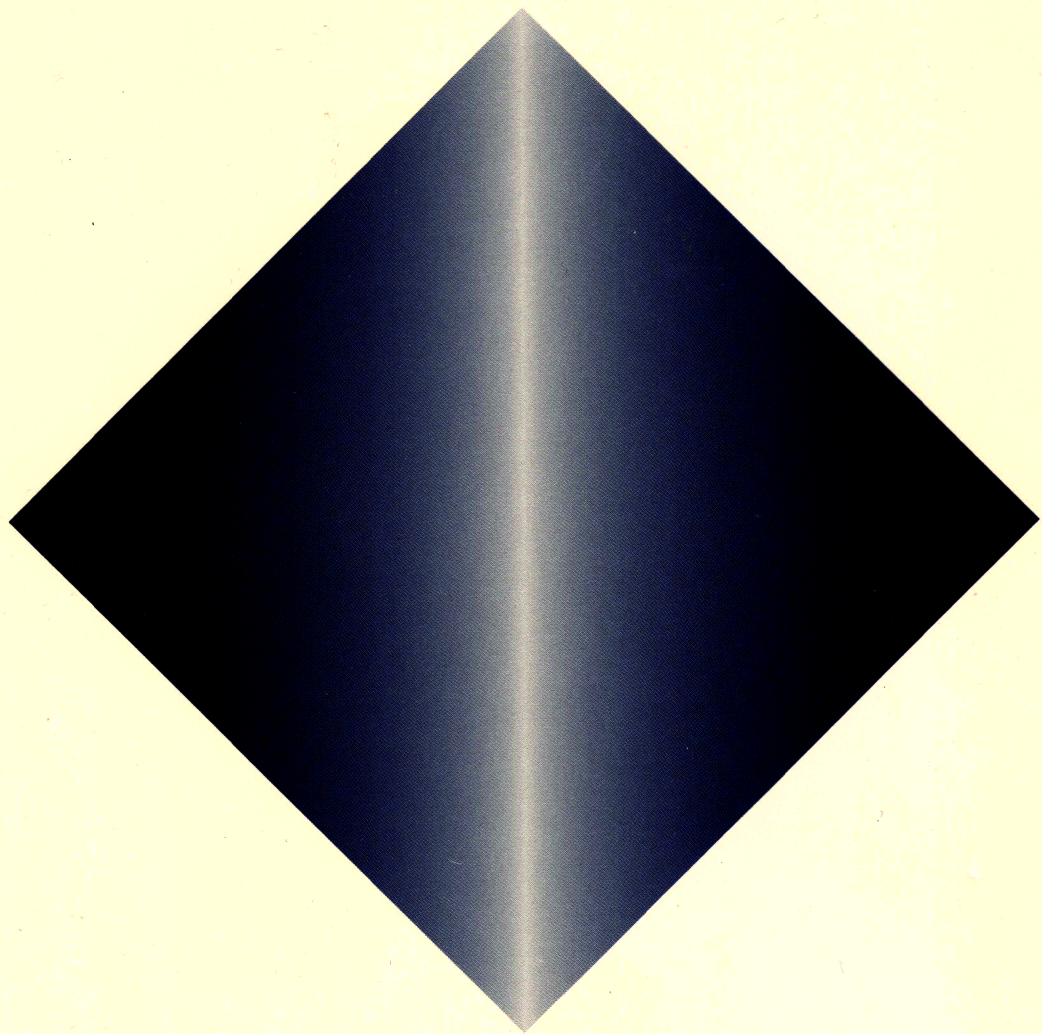


# UNIX<sup>®</sup>

SYSTEM V RELEASE 3.2



## **TCP/IP USER'S GUIDE AND REFERENCE**

REV.	REVISION	DATE
-001	Original Issue.	03/90

Additional copies of this manual or other Intel literature may be obtained from:

Literature Distribution Center  
Intel Corporation  
P.O. Box 7641  
Mt. Prospect, IL 60056-7641

For faster service in the U.S.A. or Canada, call 1-800-548-4725.

In locations outside the United States, obtain additional copies of Intel documentation by contacting your local Intel sales office. For your convenience, international sales office addresses are located directly after the reader reply card in the back of this manual.

The information in this document is subject to change without notice.

Intel Corporation makes no warranty of any kind with regard to this material, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. Intel Corporation assumes no responsibility for any errors that may appear in this document. Intel Corporation makes no commitment to update or to keep current the information contained in this document.

Intel Corporation assumes no responsibility for the use of any circuitry other than the circuitry embodied in an Intel product. No other circuit patent licenses are implied.

Intel software products are copyrighted by and shall remain the property of Intel Corporation. Use, duplication or disclosure is subject to restrictions stated in Intel's Software License, or as defined in ASPR 7-104.9(a)(9).

No part of this document may be copied or reproduced in any form or by any means without prior written consent of Intel Corporation.

The following are trademarks of Intel Corporation and its affiliates and may be used only to identify Intel products:

Above	ICEVIEW	iPSB	Promware
ACE51	iCS	iPSC*	QUEST
ACE96	iDBP	iRMK*	Quick Erase
ACE186	iDIS	iRMX*	Quick-Pulse Programming
ACE196	iLBX	iSBC*	QueX
ACE960	im*	iSBX	Ripplemode
BITBUS	iMDDX	iSDM	RMX/80
COMMPuter	iMMX	iSXM	RUPI
CREDIT	Inboard	Library Manager	Seamless
Data Pipeline	Insite	MAPNET	SLD
ETOX	Intel*	MCS	SugarCube
GENIUS	intel*	Megachassis	UPI
i486	Intel376	MICROMAINFRAME	VLSICEL
i860	Intel386	MULTIBUS*	376
i	intelBOS	MULTICHANNEL	386
i	Intel Certified	MULTIMODULE	387
i*	Intelelevision	ONCE	4-SITE
I2ICE	iOSP	OpenNET	486
ICE	iPAT	QTP	860
iCEL	iPDS	PROMPT	

XENIX, MS-DOS, Multiplan, and Microsoft are trademarks of Microsoft Corporation. UNIX, OPEN LOOK, and Documenter's Workbench are registered trademarks of AT&T. ETHERNET is a trademark of Xerox Corporation. Centronics is a trademark of Centronics Data Computer Corporation. Chassis Trak is a trademark of General Devices Company, Inc. VAX and VMS are trademarks of Digital Equipment Corporation. Smartmodem 1200 and Hayes are trademarks of Hayes Microcomputer Products, Inc. IBM and PC AT are registered trademarks of International Business Machines. MDS is an ordering code only and is not used as a product name or trademark. MDS is a registered trademark of Mohawk Data Sciences Corporation. X Window System is a trademark of Massachusetts Institute of Technology. Crystal/Writer is a registered trademark of Syntactics Corporation. ZP/ix is a registered trademark of Phoenix Technology. WEITEK is a registered trademark of MicroWay.

Copyright © 1990, Intel Corporation. All Rights Reserved.  
Copyright © 1987, 1988, 1989 Lachman Associates, Inc.  
Copyright © 1987 Convergent Technologies, Inc.  
Copyright © 1987 Sun Microsystems, Inc.

# **SYSTEM V/386 TCP**

## **USER'S GUIDE AND REFERENCE**

### **CONTENTS**

- Chapter 1 USING NETWORK COMMANDS
- Chapter 2 REMOTE COMMAND EXECUTION
- Chapter 3 REMOTE TERMINALS
- Chapter 4 FILE TRANSFER
- Chapter 5 GLOSSARY
- Chapter 6 USER'S REFERENCE

UNITED STATES DEPARTMENT OF THE INTERIOR

BUREAU OF LAND MANAGEMENT

WATER RESOURCES DIVISION

WATER RESOURCES DIVISION

WATER RESOURCES DIVISION

WATER RESOURCES DIVISION

WATER RESOURCES DIVISION

# **Chapter 1**

## **USING NETWORK COMMANDS**

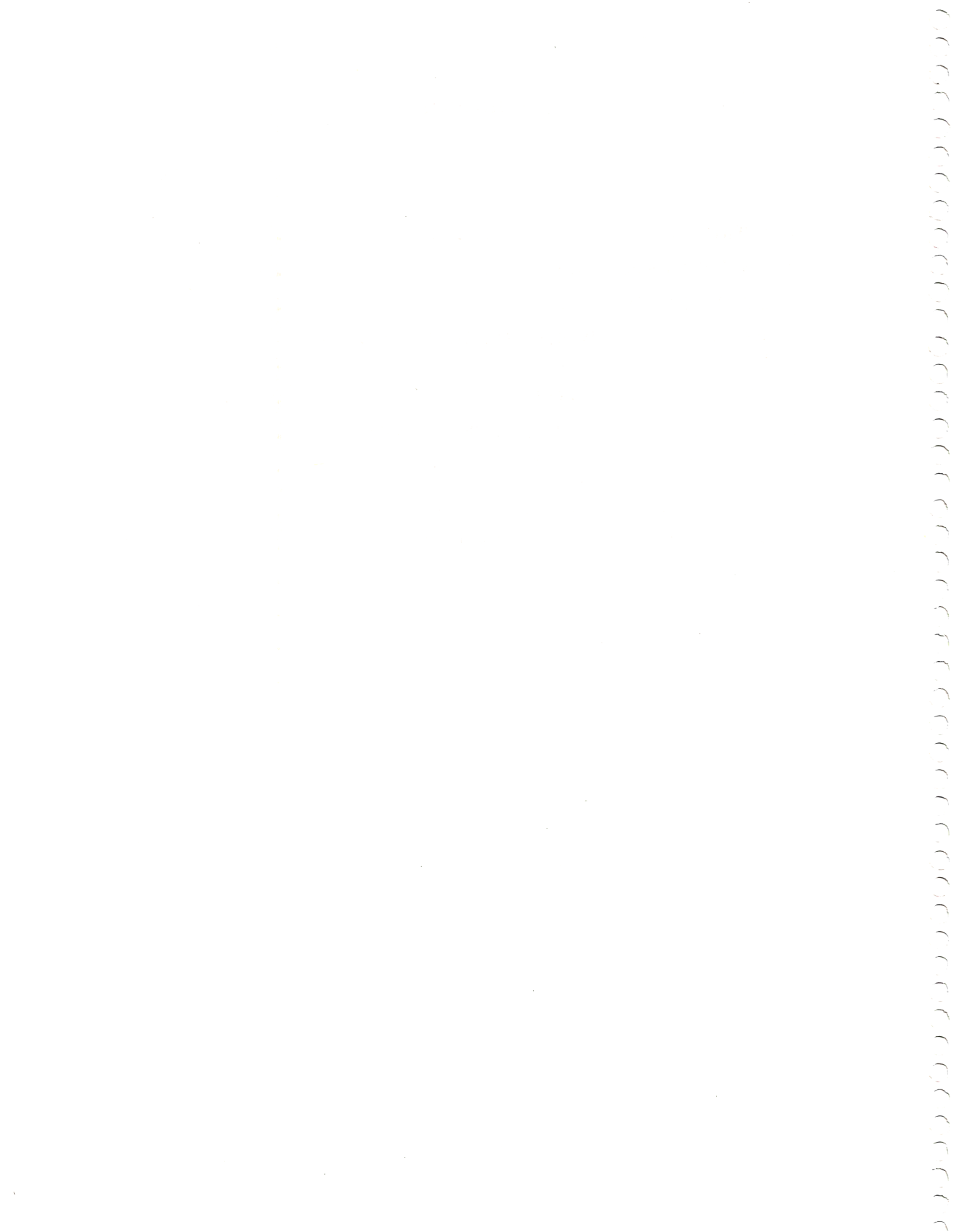
### **INTRODUCTION**

THE UNIVERSITY OF CHICAGO

PHYSICS DEPARTMENT

# Chapter 1

	PAGE
1. INTRODUCTION . . . . .	1
2. ABOUT THIS SECTION . . . . .	2
2.1 OVERVIEW . . . . .	2
3. WHAT INTERNETWORKING PROVIDE THE UNIX USER . . . . .	3
4. UNIX NETWORKING OBJECTS . . . . .	4
5. OVERVIEW OF UNIX NETWORKING COMMANDS . . . . .	5
5.1 WHAT IS USER EQUIVALENCE? . . . . .	5
5.1.1 CONNECTIONS, NAMES AND ADDRESSES . . . . .	6
5.1.2 ACCESS AND PASSWORD PROBLEMS . . . . .	6
6. VIRTUAL TERMINALS AND REMOTE LOGIN . . . . .	8
6.1 TELNET(1) . . . . .	8
6.2 REMOTE LOGIN (rlogin) . . . . .	8
7. TRANSFERRING FILES (ftp & rcp) . . . . .	9
8. REMOTE COMMAND EXECUTION (rcmd) . . . . .	10



# Chapter 1

## USING NETWORK COMMANDS

### INTRODUCTION

#### 1. INTRODUCTION

This chapter is an overview of UNIX internetworking commands for the network user and for the new administrator or programmer. You can use this chapter as a guide to the other three chapters in this section.

Some of the subjects discussed in this chapter include:

- UNIX network object types
- network commands
- use of a virtual terminal
- transferring files to and from remote machines
- remote command execution (*rcmd*)
- remote printing
- using pipes and shell scripts
- user and machine equivalences and passwords

## 2. ABOUT THIS SECTION

Chapters 2 through 4 provide detailed explanations and examples of the networking commands used for common network user functions. You can use and reference these chapters as needed, but they are intended only to supplement the coverage of the commands found in the appropriate *UNIX Operating System Manual*. (See the note below.)

Chapter 2, "Remote Execution" explains how to execute commands on other systems across the network using the command *rcmd*.

Chapter 3, "Remote Terminals" explains in detail the virtual terminal commands, *telnet*(1) and *rlogin*(1).

Chapter 4, "File Transfer," explains how to use the file transfer commands, *ftp*(1) and *rcp*(1).

### 2.1 OVERVIEW

UNIX is a command-oriented operating system. To make use of the remote resources in a UNIX internetworking environment, the user invokes network-specific commands. These commands are fully integrated with UNIX and may be invoked from the shell command line and shell programs or executed from within user programs with the *fork*(2) or *exec*(2) system calls, or the *system*(3) library routine.

These network commands are user processes of the operating system which require network software to function. In UNIX the name of the network command is the same as the name of the file that contains the process program.

---

#### - Please Note -

---

The treatment of networking commands in this section (and the rest of the chapter) is not intended to be the definitive reference source for network commands. Refer to the man pages in the reference section found at the end of this manual for details on each command.

### 3. WHAT INTERNETWORKING PROVIDE THE UNIX USER

A UNIX network based on Ethernet provides a means of linking a large number of UNIX machines so that the network user on any one of these machines can access resources and data on any of the other machines. A UNIX internet is two or more networks, possibly using a variety of machine types, protocols, and media, welded together in a flexible manner to form a larger network. The internetworking linkage is invisible at the command interface level so that the system appears to the network user as a single network.

Some of the many things you can do as a network user whose machine is connected in a UNIX network are as follows:

- Log onto another machine on which you have an account
- Move logically from one remote machine to another without having to enter your password (if your system administrators have "equated" the machines or if you have created a user equivalence for that machine)
- Execute commands on any machine in the network
  - you can execute commands where the data is (thus avoiding the moving of files)
  - you can execute commands where the load is lowest
  - you can construct sequences of UNIX commands including *pipes* which move data between machines for processing. For example:

```
pr -f myfile | rcmd grumpy lp
```

will send the output of the pr command to the printer spooler on the remote system called "grumpy".

- Access public data from all machines
- Copy or transfer files from one machine to another
- Share remote devices such as printers and tape drives
- Access electronic mail systems that have been implemented for the network
- Run applications resident on other machines
- Access other machines that are running the appropriate communications protocol.

## 4. UNIX NETWORKING OBJECTS

There are three types of UNIX networking objects:

- executable commands, and server programs (sometimes called *daemons*) supporting the commands
- configuration files
- library and system calls for use by programmers

All these types are documented in the reference manuals. User commands, configuration file formats, and library interfaces are all listed alphabetically within sections. The section number of the entry is given in parenthesis after the name of the command. Following UNIX convention,

Section 1	contains user commands
Section 1M	contains administrative commands
Section 2	contains system calls
Section 3	contains library routines
Section 4	contains file formats
Section 5	contains miscellaneous
Section 7	contains special files and protocols

For example:

*ftp*(1)

tells us that ftp is a user command that is documented in reference section 1.

## 5. OVERVIEW OF UNIX NETWORKING COMMANDS

Included in the UNIX commands are a set of commands often referred to in a Berkeley UNIX environment as the "r-commands", which are designed to be UNIX-specific. The r in r-command stands for "remote."

Another set of commands, such as *telnet* and *ftp*, originated from ARPANET. They are designed to be operating-system independent. The protocols used in these commands are specified by the DoD Internet specification.

The major difference between these two different types of commands is that the r-commands propagate UNIX-style permissions across the network. The ARPANET commands do not understand the UNIX permissions.

The networking commands are listed alphabetically in the table below with a brief description. Not all UNIX networking commands are intended for use by the network user. Some are network administrative functions.

UNIX Networking Commands	
Command	Description
ftp(1)	file transfer program
ifconfig(1M)	configure network interface parameters (administrative)
mkhosts(1M)	make node name commands (administrative)
netstat(1M)	show network status (administrative)
rcmd(1)	remote shell command execution ("rsh" in Berkeley UNIX)
rcp(1)	remote file copy
rlogin(1)	remote log in
ruptime(1)	display status of nodes on local network
rwho(1)	who is logged in on the local network nodename
slattach,	attach and detach serial lines (administrative)
sldetach(1M)	as network interfaces (administrative)
telnet(1)	user interface to DARPA TELNET protocol
trpt(1M)	print protocol trace (administrative)

### 5.1 WHAT IS USER EQUIVALENCE?

User equivalence is an existing statement on a local machine to the effect that a particular user on a remote machine is equivalent to a user by the same or a different name on the local machine and has the exact same privileges as the existing local user. The equivalent user does not need a password to log in when he uses a program that understands user equivalence. Implicit in this equivalence is that the remote user now has password

privileges on the local machine.

The remote user still needs a separate account and password set up on the remote machine. The equivalent user can use the same name on both machines or a different name.

Note that you need to have an equivalence set up for your own user name even on your local machine. If you pipe to another machine (see *sh(1)* in the UNIX User's Reference or *rcp(1)* in the TCP User's Reference), you will need an equivalence to that machine.

### 5.1.1 CONNECTIONS, NAMES AND ADDRESSES

From the perspective of the user, most internet protocols are connection-oriented. This means that for information to be communicated between your machine and a remote machine over the internet you must first have established a connection to that machine. Establishing a connection is similar to dialing a phone number when making a phone call; it defines the parties in the call and sets up a connection between them.

Although the data sent over the connection is packet-switched, rather than circuit switched as in the telephone system, the functions are alike. TCP performs the mechanics of establishing connections for you, but in many cases, *telnet* and *ftp* in particular, you have to be aware of connections and give commands to get them established.

As with dialing a phone, you must first know how to reach the recipient of your call when setting up a connection. Each host on the internet has a unique address, like a phone number, by which it can be "called" in establishing a connection. Because network addresses are not always easy to remember, the internet software allows for the use of names instead of addresses. Host names are established by your system administrator who should tell you the names of the hosts with which you may communicate. Since hosts may be used for several purposes, it is possible to have several names (aliases) for the same host address. However, each name always stands for a single host address and will connect you to the same host each time you use it.

### 5.1.2 ACCESS AND PASSWORD PROBLEMS

Often in an internetworking environment, different host machines are under the jurisdiction of different departments and personnel. Those in charge of a

host machine often wish to limit access to their own machine for various security and procedural reasons. Privileges to a machine can be given only from the machine in question. If you are unable to access a machine you have a need for, you or your supervisor can see the network administrator of the host machine you wish to access.

If you need access beyond "anonymous *ftp*", the administrator can set up a machine or user equivalence between your native host and the remote host. You will need the an account and password and on the remote machine. If you have an account on a remote machine, you can set up a user equivalence yourself.

## 6. VIRTUAL TERMINALS AND REMOTE LOGIN

The command *rlogin*(1) and the ARPANET command *telnet*(1) provide the user with a choice of virtual terminal capability. A virtual terminal is created when the user on one machine logs onto another machine and presents his terminal as being logically on that machine. Between UNIX-compatible machines, switching your terminal between machines can be as easy as typing the name of the machine to which you wish to connect.

Virtual terminal capability differs from remote command execution in that the user can use programs that depend on accessing the terminal directly, such as *vi*. Commands like *i* use the terminal in raw mode, that is, they read from the terminal character by character, instead of line by line.

The following is a brief overview of *telnet* and *rlogin*. For more information on these commands, see the chapter on "Remote Terminals" and the corresponding man pages in the reference section of this manual.

### 6.1 TELNET(1)

The *telnet* command provides virtual terminal access to other machines on the internet, even non-UNIX systems. Using *telnet*, you can log in to any host on the network for which you have an account just as if you were a local user of that machine. Once *telnet* is invoked, your terminal is linked to a remote machine and data that you type is passed to that machine. Responses from the remote machine will be displayed on your terminal's screen.

For more information on *telnet*, see the "Remote Terminals" chapter.

### 6.2 REMOTE LOGIN (rlogin)

The virtual terminal command, *rlogin*, allows the user to remotely log into another UNIX-compatible machine. This command requires a password on the host you are logging into unless you have user equivalence on that machine. The command, *rlogin*, is most suitably used when you are connecting with a UNIX-compatible host.

For more information on *rlogin*, see the "Remote Terminals" chapter.

## 7. TRANSFERRING FILES (*ftp* & *rcp*)

The ARPANET command, *ftp*, allows a user to manipulate files on two machines simultaneously. You can examine directories and move single or multiple files between systems. This program is designed to be highly independent of the operating system.

An additional feature of *ftp* is that it allows an anonymous user who does not have an account on your machine to pick up or deposit certain files from a protected area of the *ftp* home directory. *Ftp* does not require (or understand) user equivalence.

The remote file copy command, *rcp*, does require user equivalence. The *rcp* command is a UNIX-specific command and is most suitably used when you are transferring files between UNIX compatible hosts.

For more information on *ftp* and *rcp*, see the "File Transfer" chapter.

## 8. REMOTE COMMAND EXECUTION (*rcmd*)

The *rcmd* command allows you to send commands to remote UNIX machines for execution and have the results returned to you. To use *rcmd* you do not have to log onto the remote machine. (It is like a pipe to another machine.) This command is useful for constructing distributed shell programs. To use *rcmd*, the user must have equivalence on the target machine (the remote machine on which the user is trying to execute the command).

This command may only be used with remote machines running UNIX or a compatible operating system. *Rcmd* passes its standard input and outputs to the remotely executed command, and returns to the issuing system all output which the remote command generates on standard output and standard error.

You must have **/usr/hosts** in your search path to access machines directly. (For more information, see the entry for *rcmd*(1) in the TCP User's Reference.)

## **Chapter 2**

# **REMOTE COMMAND EXECUTION**

## **RCMD**

Chapter 2

THEORY OF CONFORMAL MAPPINGS

GROUP

# Chapter 2

	PAGE
1. INTRODUCTION . . . . .	1
2. REMOTE COMMAND EXECUTION . . . . .	2
2.1 INVOKING RCMD . . . . .	2
2.2 SAMPLE RCMD SESSION . . . . .	3
2.3 REMOTE PRINTING . . . . .	3
2.4 SHELLSCRIPT PROGRAMMING USING RCMD . . . . .	3



## **Chapter 2**

# **REMOTE COMMAND EXECUTION RCMD**

### **1. INTRODUCTION**

This chapter is an overview of remote command execution over the network and is written for the network user.

## 2. REMOTE COMMAND EXECUTION

The *rcmd* command allows you to send commands to remote UNIX machines for execution and have the results returned to you. To use *rcmd* you do not have to log onto the remote machine. (It is like a pipe to another machine.) This command is useful for constructing distributed shell programs. To use *rcmd*, the user must have equivalence on the target machine (the remote machine on which the user is trying to execute the command).

This command may only be used with remote machines running UNIX or a compatible operating system. *Rcmd* passes the command its standard input and outputs the command's standard output and standard error.

### 2.1 INVOKING RCMD

*Rcmd* is invoked from the UNIX shell. You must specify the name of a remote machine and one or more commands to be executed, for example,

```
$ rcmd admin ls
```

will execute the *ls* command on the machine named *admin*.

In most cases, you may omit specifying *rcmd* to the shell and simply put the name of the remote machine and a command, for example,

```
$ admin ls
```

In order for this to work, you must have */usr/hosts* in your search path to access machines directly. (For more information, see the entry for *rcmd(1)* in the TCP User's Reference chapter at the end of this manual.) The system administrator must set up the */usr/hosts* file before you can do remote command execution by typing the system name without the *rcmd*. See *mkhosts(1M)* in the TCP Administrators Reference for detailed information.

Also, you may specify two options when invoking *rcmd*:

**-l user**                      Generally, the command you specify will be executed under your user name on the remote machine. The **-l** option allows you to specify that the command be executed under another user name, for example,

```
$ rcmd admin -l tom ls
```

Whether you use your user name or another user name, you have to establish permission for yourself on the remote machine before the command can be executed.

**-n**                The **-n** option prevents *rcmd* from sending the standard input file to the remote command you specify and prevents *rcmd* from reading up the standard input file by making its standard input **/dev/null** instead of *rcmd*'s standard input. For example,

```
$ rcmd admin -n -l tom ls
```

"Reading up" means reading the file and buffering it. *Rcmd* buffers data in the standard input file regardless of whether the remote command reads it.

## 2.2 SAMPLE RCMD SESSION

The following example shows *rcmd* being used to run the *who*(1) command on a remote machine called "admin" and to place the output in a file on the local machine by redirecting standard output.

```
$ rcmd admin who > /tmp/admin.who
```

To redirect the standard output to the remote host, we would do the following:

```
$ rcmd admin who > /tmp/admin.who
```

## 2.3 REMOTE PRINTING

*Rcmd* can be used for remote printing, as in the following which prints file **testfile** on the default printer on system **systemx**:

```
$ cat testfile | rcmd systemx lp
```

## 2.4 SHELLSCRIPT PROGRAMMING USING RCMD

Many useful shell programs can be written using the capabilities of the UNIX networking commands to use pipes across the network. (See *pipe*(2) in the UNIX Programmer's Reference Manual.) Such shell programs can be the glue that make a distributed system most useful. Some examples of systems based on shell programs are:

- remote line printer spooling using *rcmd* and the UNIX *lp* system.
- distributed text processing using *troff*(1). In this system, macroprocessing is done at the user's node, the font crunching is done on a lightly loaded back-end machine, and printing is done on a machine with a laser printer.

- read/write a cpio archive on a remote tape drive
- kill a process on a remote machine
- backup/restore remote file systems

## **Chapter 3**

### **REMOTE TERMINALS**

#### **TELNET and RLOGIN**

Chapter 1

RECENT RESEARCH

THEORY AND RESEARCH

## Chapter 3

	PAGE
1. Introduction . . . . .	1
2. THE TELNET COMMAND . . . . .	2
2.1 COMMAND AND INPUT MODES . . . . .	2
2.2 TELNET OPTIONS . . . . .	2
2.3 INVOKING TELNET . . . . .	2
2.4 TELNET COMMANDS . . . . .	3
2.5 SAMPLE SESSIONS . . . . .	7
2.5.1 Description of Session 1 . . . . .	7
2.5.2 Description of Session 2 . . . . .	8
3. THE RLOGIN COMMAND . . . . .	10
3.1 INVOKING RLOGIN . . . . .	10
3.2 RLOGIN OPTIONS . . . . .	10
3.3 USING A TILDE IN THE TEXT . . . . .	11
3.4 EXITING RLOGIN . . . . .	11



## Chapter 3

# REMOTE TERMINALS TELNET and RLOGIN

### 1. Introduction

This chapter explains two commands that provide virtual terminal capability. "Terminal" indicates that the command allows your terminal on your local machine to act as a terminal on a remote machine over the internet. "Virtual" indicates that no physical connection is made to the remote machine. Rather, the command simulates a physical line between your terminal and a remote machine.

The virtual terminal commands are

- *telnet*(1)
- *rlogin*(1)

The *telnet* command provides virtual terminal access to other machines on the internet. Using *telnet*, you can login to any host on the network for which you have permission just as if you were a local user of that machine. Once *telnet* is invoked, your terminal is linked to a remote machine and data that you type is passed to that machine. Responses from the remote machine will be displayed on your terminal's screen.

For communicating with other machines running the UNIX operating system, the *rlogin* command can be used in place of *telnet*. *Rlogin* provides a virtual terminal access to UNIX-like machines that is specific to the UNIX operating system. See "The Rlogin Command," below.

## 2. THE TELNET COMMAND

Telnet is an interactive program which allows you to communicate with a remote machine in a terminal session. Once you invoke *telnet*, you will interact with *telnet* until you exit and return to the shell (calling program).

### 2.1 COMMAND AND INPUT MODES

Whenever *telnet* is connected to a remote machine, it operates in input mode. Input mode transfers all the characters you type to the remote machine and displays all data sent to you by the remote machine on your terminal's screen. The one exception to this is a special character called the escape character, `^]`, which places *telnet* in command mode if you type it. (This escape character is not the same as the **Escape** command of your keyboard. It is produced by typing **Control-]**).

In command mode, data that you type is interpreted by *telnet* to allow you to control *telnet* operation. Command mode is also active when *telnet* is not connected to a remote host.

### 2.2 TELNET OPTIONS

When *telnet* is in input mode, it communicates with the remote host based on a number of options. These options specify how operating system and terminal specific properties of terminal to computer communications, such as whether the echoing of the characters you type is done by *telnet* locally or by the remote machine, will be performed. *Telnet* and the remote machine you specify will negotiate these options and establish a compatible set of options for your terminal when you connect to a host.

### 2.3 INVOKING TELNET

You invoke *telnet* from the UNIX shell with the command *telnet*.

Optionally, you may specify the name of the remote machine with which you wish to communicate. For example:

```
telnet admin
```

Machine names are defined by your system administrator. You can examine the machine names available to you by listing the contents of the file `/etc/hosts`.

When you specify a machine name when you invoke *telnet*, *telnet* will establish a network connection to that machine and enter input mode. You may also invoke *telnet* without a machine name, for example:

```
telnet
```

you do not specify a machine name, you must open a connection from within *telnet* using *telnet's* **open** command before you can log into a remote host. See "Telnet Commands" below.

## 2.4 TELNET COMMANDS

You may enter *telnet* commands whenever the *telnet* command mode prompt is displayed. The *telnet* command mode prompt looks like:

```
# telnet
telnet>
```

Telnet will be in command mode if you are not connected to a remote machine or when you enter the escape character from input mode.

If command mode was not entered from input mode, *telnet* will generally remain in command mode and display the command mode prompt again after you enter each command. If you use the **open** command to establish a *telnet* connection to a remote machine, *telnet* will enter input mode.

If command mode is entered from input mode, *telnet* generally will return to input mode after processing your command. If you use the **close** command to close the remote host connection, *telnet* will remain in command mode after the command is processed. If you use the **quit** command, *telnet* will exit and return you to the calling program, usually the shell.

Each command you give to *telnet* in command mode must be followed by a **Return**. *Telnet* will not start a command until it receives a **Return** from you. If you make a mistake while typing a command, you may use the shell line editing commands erase (**Backspace**) and kill (**Cancel**) to edit the characters that you have typed.

When entering a command, you do not have to enter the full command name. You need only enter enough characters to distinguish the command from other *telnet* commands.

Also, please note that the definitive syntax for all *telnet* commands should be found on the manual page *telnet(1)* in the UNIX User's Reference Manual.

open	<p>This command establishes a <i>telnet</i> connection to a remote machine. You should specify the name of the remote machine as an option of the command, for example,</p> <pre>telnet &gt; open admin</pre>				
close	<p>This command closes the connection to the remote host and causes <i>telnet</i> to enter command mode.</p>				
quit	<p>This command terminates your <i>telnet</i> session and exits <i>telnet</i>. The quit command closes the connection to the remote machine if one is active.</p>				
z	<p>Suspend <i>telnet</i>. On systems with job control, this suspends <i>telnet</i>. On other systems, it provides the user with another shell.</p>				
mode	<p>The following are subcommands/options of the <b>mode</b> command, whose syntax is described in the man page <i>telnet(1)</i>:</p> <pre>mode [ line   character ]</pre> <table> <tr> <td>line</td><td>The remote host is asked for permission to go into line at a time mode.</td></tr> <tr> <td>character</td><td>The remote host is asked for permission to go into character at a time mode.</td></tr> </table>	line	The remote host is asked for permission to go into line at a time mode.	character	The remote host is asked for permission to go into character at a time mode.
line	The remote host is asked for permission to go into line at a time mode.				
character	The remote host is asked for permission to go into character at a time mode.				
display	<p>Displays all or some of the <i>set</i> or <i>toggle</i> values.</p>				
send	<p>Sends one or more special character sequences to the remote host. The subcommands/options of the <b>send</b> command are fully described in the man page <i>telnet(1)</i>:</p> <pre>send [ ao   ayt   brk   ... ]</pre> <table> <tr> <td>ao</td><td>This command causes <i>telnet</i> to tell the remote machine to abort sending any output that is in progress. This command is useful if the remote host is sending you data that you do not wish to see and you would like <i>telnet</i> to return to command mode on the remote machine. The only output aborted is that currently being sent, you may continue to communicate with the remote machine once the current output has been stopped.</td></tr> <tr> <td>ayt</td><td>This command causes <i>telnet</i> to send an "are you there?" message to the remote machine.</td></tr> </table>	ao	This command causes <i>telnet</i> to tell the remote machine to abort sending any output that is in progress. This command is useful if the remote host is sending you data that you do not wish to see and you would like <i>telnet</i> to return to command mode on the remote machine. The only output aborted is that currently being sent, you may continue to communicate with the remote machine once the current output has been stopped.	ayt	This command causes <i>telnet</i> to send an "are you there?" message to the remote machine.
ao	This command causes <i>telnet</i> to tell the remote machine to abort sending any output that is in progress. This command is useful if the remote host is sending you data that you do not wish to see and you would like <i>telnet</i> to return to command mode on the remote machine. The only output aborted is that currently being sent, you may continue to communicate with the remote machine once the current output has been stopped.				
ayt	This command causes <i>telnet</i> to send an "are you there?" message to the remote machine.				

The remote machine will send you a message back if it is active. This message is often simply causing the bell on your terminal to sound although it may be a string of text which is displayed on your terminal. This message is useful if the remote host has not responded to your input and you wish to see if it is inactive or just busy.

**brk**

This command sends a message to the remote machine which has the same significance as pressing the **Break** key on your terminal would to your local machine. Since **brk** is implemented between a terminal and a local machine as a set of physical signals, rather than data, pressing the **Break** key on your terminal affects only the local machine and is not sent to the machine to which you are connected via *telnet*. You must use the **brk** command if you want to send a break indication to a remote machine.

**ec**

This command sends the *telnet* erase character message to the remote machine. **ec** has the same meaning as the shell erase (backspace) command does on your local machine. Since different operating systems implement the erase character operation differently, you may have to use the **ec** command, rather than the shell erase character, when interacting with a remote machine. The shell erase character can be used in command mode since command mode's operation is local to your machine.

**el**

This command sends the *telnet* erase line message to the remote machine. **EL** has the same meaning as the shell kill (erase line) command does on your local machine. Since different operating systems implement the erase line operation differently, you may have to use the **EC** command, rather than

the shell kill command, when interacting with a remote machine.

The shell kill command may be used in command mode since command mode's operation is local to your machine.

**ip**

This command sends the *telnet* interrupt process message to the remote machine. **IP** has the same meaning as the shell interrupt command does on your local machine. Since different operating systems implement the interrupt operation differently, you must use the **IP** command, rather than the shell interrupt command, when interacting with a remote machine. The shell interrupt command may be used in command mode since command mode's operation is local to your machine.

**synch**

This command sends a message to the remote machine telling it to ignore any input you have sent but which has not yet been processed on the remote machine. This command is useful if you have typed ahead a number of commands and wish to cancel these commands without terminating the *telnet* connection to the remote machine.

**escape**

Sends the current *telnet* escape character.

**nop**

Sends the *telnet* no-operation sequence.

**toggle**

Toggles various flags which control *telnet* processing between **TRUE** and **FALSE**. The subcommands/options of the **toggle** command are fully described in the man page *telnet(1)*:

```
toggle [ localchars | autoflush | ... ]
```

**set**

This command allows you change *telnet* variable values. There are subcommands/options of the **set** command, and their syntax is described in the man page *telnet(1)*:

```
set [ echo | escape | interrupt | ... ]
```

**status**

This command shows you the status of the connection to the remote host as well as the current options and escape

character.

?

This command displays information on your terminal about operating *telnet*. If you specify a command name to help, information about that command is displayed. If you just enter help, a list of all commands is displayed.

## 2.5 SAMPLE SESSIONS

A number of sample sessions are shown below which illustrate how *telnet* can be used in a variety of ways. Communications with a host named THERE are shown.

### 2.5.1 Description of Session 1

This is a simple session illustrating basic *telnet* use. *Telnet* is invoked with a host name and opens a connection to that host. *Telnet* displays "Trying..." to indicate it is trying to establish a connection and a message indicating it is connected when the connection is established. *Telnet* displays the current escape character. (There is no options status display.) At this point, *telnet* has established the connection to the remote machine and the remote machine displays its login prompt. The user then logs into the machine using the same procedures that would be used for a local terminal on that machine. The user does a listing of his directory on the remote machine. Having completed his work, the user then types the escape character and *telnet* enters command mode and displays the command mode prompt. The user enters the quit command and *telnet* closes the connection to the remote machine and returns to the local shell.

```
laiter$ telnet laioff
Trying 192.9.200.101 ...
Connected to laioff.
Escape character is '^['.
```

```
System V.3.2 UNIX (laioff.Lachman.COM)
```

```
login: stevea
```

```
Password:
```

```
UNIX System V/386 Release 3.2
```

```
laioff
```

```
Copyright (C) 1984, 1986, 1987, 1988 AT&T
```

```
Copyright (C) 1987, 1988 Microsoft Corp.
```

```
All Rights Reserved
```

```
Login last used: Mon Feb 27 17:14:18 1989
```

```
laioff$ ls -xF
```

bell/	blot/	connect.h	connection.c	dhry/
hi*	hi+.c	hi.c	hin*	hin.c
hn*	hn.c	indent/	intel/	ip_icmp.h
jam/	linger*	linger.c	mailstats.c+	maketd/
maketd+/	maxmin	ot*	ot.c	ot2*
ot2.c	ping+*	ping.c	profiler/	qt/
ripsoak*	ripsoak.c	sr.sh*	st.c	sw/
t*	t.c	tcp/	tcp.sh*	tcp0227/

```
laioff$
```

```
^]
```

```
telnet> quit
```

```
Connection closed.
```

```
laiter$
```

## 2.5.2 Description of Session 2

This session illustrates alternative ways to log in and out of a remote machine with *telnet*. *Telnet* is invoked without a machine name and enters command mode. The user does a status command and *telnet* indicates that no connection is established. The user then uses the *telnet open* command to establish a connection and place *telnet* into input mode. The user receives a login message from the remote system. The user then logs into the machine using the same procedures that would be used for a local terminal on that machine. Having completed his work, the user logs out of the remote machine. The remote machine then closes the connection. *Telnet* terminates automatically and returns to the local shell.

```
laiter$ telnet cliff
Trying 192.9.200.26 ...
Connected to cliff.
Escape character is '^']'.
```

System V.3.2 UNIX (cliff.Lachman.COM)

```
login: aes
Password:
UNIX System V Release 3.2
cliff
Copyright (c) 1987 AT&T
ALL RIGHTS RESERVED
Login last used: Thu Feb  2 11:26:03 1989
```

```
/          :      Disk space:   4.70 MB of  14.94 MB available (31.50%) .
/usr       :      Disk space:   5.29 MB of  45.65 MB available (11.61%) .
```

Total Disk Space: 10.00 MB of 60.59 MB available (16.51%) .

```
[cliff] cd /usr/tcptest
[cliff] ls -xF
Config*      NConfig      README      bint/       data/
ftpsoak.out  run.out      src/
[cliff] exit
Connection closed by foreign host.
laiter$
```

### 3. THE RLOGIN COMMAND

The *rlogin*(1) command connects you to a shell on a remote machine. *Rlogin* is similar to *telnet* but is specific to UNIX-compatible machines and allows you to access the same UNIX commands on a remote machine as *telnet* but is more convenient than *telnet* in that, once you have logged onto a remote machine, it is as if it is now your local machine and you do not have to know the special commands used in *telnet*. This command can only be used with remote machines running UNIX or a compatible operating system. The TERM variable in the remote shell is set to the value you are using in your local shell.

Once invoked, *rlogin* will pass all data you input to the remote machine and display all output from that machine on your terminal's screen.

#### 3.1 INVOKING RLOGIN

Rlogin is invoked from the UNIX shell. You must specify the name of a remote machine, for example,

```
rlogin admin
```

In most cases, you may omit specifying *rlogin* to the shell and simply put the name of the remote machine, for example,

```
admin
```

Your system administrator must have configured UNIX to accept the name of the remote machine without specifying *rlogin* in order for you to be able to use this feature. You must also have */usr/hosts* in your search path. Your system administrator can advise you on how your machine is configured.

#### 3.2 RLOGIN OPTIONS

You may specify three options when invoking *rlogin*.

**-ec**                The **-e** options causes *rlogin* to use the character *c* instead of tilde (~) as the escape character to enter when exiting *rlogin*, for example,

```
rlogin admin -e!
```

sets the exclamation point as the *rlogin* escape character.

**-l <user>**        The **-l** option (lower-case L) allows you to specify that you wish to be logged in under another user name, for example,

```
rlogin admin -l mike
```

(Generally, *rlogin* logs you in to the remote machine with the same user name as you are using on your local machine.)

-8           The -8 option tells rlogin to turn off the stripping of parity bits, and pass 8 bit characters through to the remote end.

Whether you use your user name or another user name, you must have established user equivalence for yourself on the remote machine to which you are logging in. The system administrator of the remote machine can advise you on how the remote machine is configured.

### 3.3 USING A TILDE IN THE TEXT

To send a line of input beginning with a tilde (~) to the remote machine, begin that line with another tilde (the escape character).

### 3.4 EXITING RLOGIN

To exit *rlogin* and return control to your local shell, type the escape character (the tilde) and a period (~.).

Simply exiting your remote shell also causes *rlogin* to return control to your local shell.



## **Chapter 4**

### **FILE TRANSFER**

#### **FTP and RCP**

Chapter 4

THE LIBRARY

THE LIBRARY

## Chapter 4

	PAGE
1. Introduction . . . . .	1
2. OVERVIEW . . . . .	2
3. THE FTP COMMAND . . . . .	3
3.1 COMPATIBILITY OF FTP COMMANDS WITH INTERNET SYSTEMS . . . . .	3
3.2 FTP FILE TRANSFER MODES . . . . .	3
3.3 FTP FILE NAMING CONVENTIONS . . . . .	3
3.4 INVOKING FTP . . . . .	4
3.4.1 Ftp Command Options . . . . .	4
3.4.2 Using the .Netrc File For Automatic Login . . . . .	6
3.5 FTP COMMAND DESCRIPTIONS . . . . .	7
3.6 SAMPLE FTP SESSIONS . . . . .	16
3.6.1 Description of Session 1 . . . . .	16
3.6.2 Description of Session 2 . . . . .	17
4. UNIX FILE COPY - THE RCP COMMAND . . . . .	19
4.1 INVOKING RCP . . . . .	19
4.2 OPTIONS TO RCP . . . . .	20
4.3 SAMPLE RCP SESSIONS . . . . .	20



# Chapter 4

## FILE TRANSFER

### FTP and RCP

#### 1. Introduction

This chapter describes two command programs you can use to transfer files, *ftp*(1) and *rcp*(1). Information in this chapter includes:

- when and why to use the commands, including sample sessions
- how to invoke and exit the commands
- how to use the command options
- detailed descriptions of the commands you can use within the *ftp* program
- description of the *rcp* program

## 2. OVERVIEW

The *ftp* command allows you to transfer files between your current node and other machines on the internet. *Ftp* is an ARPANET command program. *Ftp* is an interactive program which allows you to input a variety of commands for file transmission and reception, and for examining and modifying file systems of machines on the network. Once you invoke *ftp*, you interact with *ftp*'s command mode until you exit *ftp* and return to the calling program.

Once *ftp* is invoked, a set of commands is provided for use within *ftp*. These are described below in alphabetical order under "FTP Commands."

*Ftp* is available under a wide range of operating systems. When communicating with machines running the UNIX operating system, the *rcp* command can be used in place of *ftp*. *Rcp* provides file transfer among UNIX machines that is specific to the UNIX operating system.

### 3. THE FTP COMMAND

#### 3.1 COMPATIBILITY OF FTP COMMANDS WITH INTERNET SYSTEMS

In addition to *ftp* commands that use standard *ftp* protocol functions, a number of commands are provided that use optional *ftp* protocol functions that cannot be supported by all operating systems. These commands should be used only in communicating with machines running UNIX or a compatible operating system. The commands whose use should be restricted in this way are indicated in the command descriptions, below. When communicating with a remote machine that does not run UNIX, you should ask your system administrator whether it supports these *ftp* commands. Some *ftp* servers do not support all the commands.

#### 3.2 FTP FILE TRANSFER MODES

Ftp allows you to transfer files in one of two modes: ASCII mode and binary mode. ASCII mode is used for text files which can be represented in standard ASCII code. Binary mode is used for binary data which must be represented as strings of contiguous bits. For communication among UNIX machines, the ASCII mode can be used for most file transfers. For communication to non-UNIX machines, the binary mode may be required for transferring some files such as program object modules. Your system administrator can advise you on when to use which file transfer mode.

#### 3.3 FTP FILE NAMING CONVENTIONS

If the first character of a file name you specify to *ftp* is a hyphen (-), *ftp* uses its standard input (for reading) or the standard output (for writing).

If the first character of a file name you specify to *ftp* is a vertical bar (|), the remainder of the file name is interpreted as a shell command. *Ftp* will create a shell with the file name supplied as a command and then use its standard input (for reading) or the standard output (for writing). If the shell command includes spaces, the file name must be appropriately quoted. For example

```
"| ls -ls"
```

### 3.4 INVOKING FTP

You invoke *ftp* from the UNIX shell with the command *ftp*.

After *ftp* is started, the *ftp* prompt is displayed on your terminal. The *ftp* prompt looks like:

```
ftp>
```

Optionally, you may specify the name of the remote machine with which you wish to communicate. For example:

```
$ ftp admin
```

Machine names are defined by your system administrator. Before using *ftp*, you can examine the machine names available to you by listing the contents of the file */etc/hosts*.

When you specify a machine name when you invoke *ftp*, *ftp* will establish a network connection to that machine to allow you to transfer files. This is equivalent to using the *ftp* open command to start a connection to the host you name. You may also invoke *ftp* without a machine name, for example:

```
$ ftp
```

If you do not specify a machine name from the shell, you must open a connection from within *ftp* using *ftp*'s open command before you can transfer files. See "FTP Commands" for a description of the open command.

#### 3.4.1 Ftp Command Options

In addition to specifying a host name when invoking *ftp*, you may also specify a number of options which modify how *ftp* will operate. These options must be placed after the command name (*ftp*) but before the host name if you are specifying one. The options you may specify when invoking *ftp* all consist of a hyphen (-) followed by a single letter, for example, *-v*.

(Each of these options has a corresponding command, of the same name, that can be used within *ftp*. You should compare the use of the options with the corresponding *ftp* command. See "Ftp Commands" below, for a description of the *ftp* commands.)

*-v* causes *ftp* to operate in verbose mode. In verbose mode, the *ftp* protocol messages sent by the remote machine to *ftp* are displayed on your terminal. Also, if you use verbose mode, statistics are displayed after the completion of each file

transfer. Normally, verbose mode is on by default if *ftp* is being run interactively. If *ftp* is being run in a script, verbose mode is off, and the **-v** option will turn on verbose mode. You may also modify whether verbose mode information is displayed from within *ftp* with *ftp*'s verbose command.

- d causes *ftp* to operate in debug mode. In debug mode, the *ftp* protocol messages sent by *ftp* to the remote machine are displayed on your terminal. If you do not use the **-d** option, this information is not displayed. You may also modify whether debug mode information is displayed from within *ftp* with *ftp*'s debug command.
- i causes *ftp* to transfer files in image (binary) mode. If you do not use the **-i** option, files are transferred in ASCII mode. You may also modify which file transfer mode to use from within *ftp* with *ftp*'s ASCII and binary commands.
- n causes *ftp* to not use autologin mode when connecting to a remote machine. When autologin mode is used, *ftp* will try to automatically identify you to the remote machine and log you in to that machine. If you use the **-n** option to turn off autologin, you will have to use the user command to login to the remote machine manually.
- g causes *ftp* to disable expansion of UNIX file name wild cards such as \*. If you do not use the **-g** option, *ftp* will expand file names you enter with wild cards in them into lists of files. You may also modify whether wild card expansion is used from within *ftp* with *ftp*'s **glob** command.

Some examples of options:

```
$ ftp -v -d admin
```

invokes *ftp* with verbose and debug modes on and causes *ftp* to open a connection to the remote machine named admin. Debug mode causes the commands sent to the remote machine to be displayed. Verbose mode causes us to see the responses received and the statistics in bytes received.

```
$ ftp -v -d
```

invokes *ftp* with verbose and debug modes on but does not cause any connection to be opened.

```
$ ftp -n -g admin
```

invokes *ftp* with autologin and wild card expansion mode off and causes **ftp** to open a connection to the remote machine named **admin**.

```
$ ftp -n -g
```

invokes *ftp* with autologin and wild card expansion mode off but does not cause any connection to be opened.

### 3.4.2 Using the .Netrc File For Automatic Login

As an optional convenience feature you can create a file named **.netrc** in your home directory. This file contains a line entry containing the login data for each machine you wish *ftp* to open automatically. See *netrc(4)* for detailed information on this file.

When you invoke *ftp* specifying a machine, or when you subsequently *open* a machine, *ftp* reads the **.netrc** file. If you have an entry for that particular machine, *ftp* automatically conducts the login protocol exchange with its counterpart at the remote machine. It supplies your login name and password if you have entered your password in the file. If you open a machine in verbose mode, you can see the transactions taking place.

The format of the file consists of blank-separated fields introduced by keywords:

```
machine <name> login <name> password <password>
```

where machine, login, and password are keywords followed by the literal data needed for login:

machine	The name of the node.
login	The user login name for that node.
password	The user's password on that node. (The password is given in normal, unencrypted text.) If you do put your password in the file, you must read/write protect the file, by setting permissions, to prevent discovery of your password, otherwise <i>ftp</i> will not let you use the file. (There is still some risk here in putting your password in the file. You must weigh the security considerations.) Ask your system administrator before using this feature.

If you do not enter your password in the file, *ftp* prompts you for your password.

Example:

```
machine admin login guido password open
```

where "admin" is the node; "guido" is the user who logs into admin; "open" is guido's password.

### 3.5 FTP COMMAND DESCRIPTIONS

When *ftp* displays this prompt, you can enter one of the commands described below. When the command is complete, the *ftp* prompt is displayed again. Depending on whether you turn on verbose or debug modes, other messages may also appear on your terminal.

Each command you give to *ftp* must be followed by a return. *Ftp* will not start a command until it receives a return from you. If you make a mistake while typing a command, you may use the shell line editing commands *erase* and *kill* to edit the characters that you have typed.

You do not have to enter the full command name, only enough characters to distinguish the command from other *ftp* commands. In each of the following command descriptions, the minimum number of characters you are required to enter are underlined in the command name at the beginning of the description.

**!** The **!** command causes *ftp* to be suspended and a shell on the local machine to be invoked on your terminal. Any character(s) you type after entering the exclamation point are executed as a command. You can return to *ftp* by exiting the shell. This returns all *ftp* options and remote machine connections in the same state as before you gave this command.

**append** The **append** command causes *ftp* to add the contents of a local file to the end of a file on the remote machine to which you are currently connected. You may specify the files to be used when invoking the command, for example

```
ftp> append localfile remotefile
```

or you may just use the command name and have *ftp* prompt

you for the file names, for example,

```
ftp> append  
(local-file) localfile  
(remote-file) remotefile
```

When you use the `append` command, the remote machine you are connected to must be a machine running UNIX or a compatible operating system.

- ASCII** The **ASCII** command causes *ftp* to transfer files in ASCII mode.
- bell** The **bell** command causes *ftp* to sound the bell at your terminal after each file transfer is completed. The next time you enter the bell command, *ftp* will stop sounding the bell after file transfers.
- binary** The **binary** command causes *ftp* to transfer files in binary mode.
- bye** The **bye** command terminates your *ftp* session and exits *ftp*. The bye command closes all your open connections.
- cd** The **cd** command changes the directory that you are working in on the remote machine to a new directory name. You may specify the new directory name when invoking the command, for example,
- ```
ftp > cd /usr/bin
```
- or you may just use the command name and have *ftp* prompt you for the new directory, for example,
- ```
ftp> cd  
(remote-directory) /usr/bin
```
- close** The **close** command closes the current connection.
- debug** The **debug** command turns on and off debug mode. If debug mode is on, messages sent by *ftp* to the remote machine are displayed on your terminal. If debug mode is off, this information is not displayed.
- delete** The **delete** command deletes a file on the remote machine to which you are currently connected. You may specify the name of the file to be deleted when invoking the command, for example,

```
ftp> delete remotefile
```

or you may just use the command name and have *ftp* prompt you for the file name, for example,

```
ftp> delete  
(remote-file) remotefile
```

**dir**

The **dir** command displays a detailed listing of the contents of a directory on the remote machine to which you are currently connected. (Compare **ls** below.) You can specify the name of the directory to be listed when invoking the command, for example,

```
ftp> dir /usr/bin
```

If you do not specify a directory name, the current working directory on the remote machine is listed.

You can also specify that the results of this command are placed in a file rather than displayed on your terminal by giving *ftp* a file name on your local machine in which to store the directory listing, for example,

```
ftp> dir /usr/bin printfile
```

You must specify a directory name with the printfile. If you want to list the current directory in a file called "printfile," use:

```
ftp> dir . printfile
```

where "." stands for the current directory.

**form**

The **form** command displays the file format used. Currently, only the nonprint format is supported.

**get**

The **get** command retrieves a file from the remote machine to which you are currently connected and stores it on your machine. You may specify the name of a file on the remote machine and a file name on your machine for the file to be stored in when you invoke the command, for example,

```
ftp> get remotefile localfile
```

Or you can simply specify the name of a file on the remote machine to retrieve the file to your local machine and give it the same name as the file on the remote machine,

```
ftp> get remotefile
```

Or you may just use the command name and have *ftp* prompt you for the file names to use, for example,

```
ftp> get
(remote-file) remotefile
(local-file) localfile
```

If you omit the local file name, the `get` command will create a file on your machine with the same name as the file on the remote machine.

**glob** The **glob** command causes *ftp* to disable expansion of UNIX file name wild cards such as \*. The next time you enter the **glob** command, wild card expansion will be reenabled. If wild card expansion is enabled, *ftp* will expand file names you enter with wild cards in them into lists of files.

**hash** The **hash** command causes *ftp* to display a pound sign (#) after each block of data it sends to or receives from the remote host. The size of a data block varies from system to system, but is typically 1024 bytes. The next time you enter the hash command, *ftp* will stop displaying pound signs after each data block.

**help** The **help** command displays information on your terminal about operating *ftp*. If you specify a command name to help, information about that command is displayed. If you just enter help, a list of all the *ftp* commands is displayed.

**lcd** The **lcd** command changes the working directory used by *ftp* on your machine. You may specify a directory name to be used as the working directory, for example,

```
ftp> lcd /usr/deb
```

If you do not specify a directory name, your home directory will be used.

**ls** The **ls** command displays an abbreviated listing of the contents of a directory on the remote machine to which you are currently connected. You may specify the name of the directory to be listed, for example,

```
ftp> ls /usr/bin
```

If you do not specify a directory name, the current working directory on the remote machine is listed.

You may also specify that the results of this command are placed in a file rather than displayed on your terminal by giving *ftp* a file name on your local machine in which to store the directory listing, for example,

```
ftp> ls /usr/bin printfile
```

You must specify a directory name with the `printfile`. If you want to list the current directory in file called "printfile," use:

```
ftp> ls . printfile
```

where "." stands for the current directory.

#### **mdelete**

The **mdelete** command deletes a list of files on the remote machine to which you are currently connected. You may specify the name of the files to be deleted when invoking the command, for example,

```
ftp> mdelete remotefile1 remotefile2
```

or you may just use the command name and have *ftp* prompt you for the file name(s), for example:

```
ftp> mdelete  
(remote-file) remotefile1 remotefile2
```

#### **mdir**

The **mdir** command obtains a directory listing for a list of remote files and places the result in a local file. You may specify the list of remote files and the local file when invoking the command for example,

```
ftp> mdir remotefile1 remotefile2 printfile
```

or you may just use the command name and have *ftp* prompt you for the file name, for example,

```
ftp> mdir  
(remote-files) remotefile1 remotefile2 printfile  
local-file printfile? y
```

#### **mget**

The **mget** command retrieves several files from the remote machine to which you are currently connected and stores them on your machine. The files stored on your machine have the same names as the files on the remote machine.

You may specify the list of remote files when invoking the command for example,

```
ftp> mget remotefile1 remotefile2
```

or you may just use the command name and have *ftp* prompt you for the file names, for example,

```
ftp> mget  
(remote-files) remotefile1 remotefile2
```

#### **mkdir**

The **mkdir** command creates a directory on the remote machine to which you are currently connected. You may specify the name of the directory to be created when

invoking the command for example,

```
ftp> mkdir /u/mydir
```

or you may just use the command name and have *ftp* prompt you for the directory name, for example,

```
ftp> mkdir  
(directory-name) /u/mydir
```

Not all *ftp* servers support the **mkdir** command.

**mls**

The **mls** command obtains an abbreviated directory listing for a list of remote files or directories and places the result in a local file. You may specify the list of remote files or directories and the local file when invoking the command for example,

```
ftp> mls remotefile1 remotefile2 printfile
```

or you may just use the command name and have *ftp* prompt you for the file name, for example,

```
ftp> mls  
(remote-files) remotefile1 remotefile2 printfile  
local-file printfile? y
```

**open**

The **open** command establishes a connection to a remote machine which may then be used for file transfer commands. You may specify the name of the remote machine when invoking the command, for example,

```
ftp > open admin
```

or you may just use the command name and have *ftp* prompt you for the machine name, for example,

```
ftp> open  
(to) admin
```

If you specify a host name when invoking the command, you may also optionally specify a port number on the remote machine. If a port number is specified, *ftp* will attempt to open a connection to the remote machine at that port rather than the default port for *ftp*. You should only use this option if you are asked to do so by your system administrator. If you do not specify a port number, *ftp* will not prompt you for one.

**prompt**

The **prompt** command causes *ftp* not to ask you for permission to proceed between files in multiple file commands such as **mget**. The next time you enter the prompt

command, *ftp* will start asking you for permission to proceed between files.

put

The **put** command transfers a file from the local machine to the remote machine to which you are currently connected and stores it. You may specify the name of a file on your machine and a file name on the remote machine when you invoke the command, for example,

```
ftp> put localfile remotefile
```

or

```
ftp> put localfile
```

or you may just use the command name and have *ftp* prompt you for the file name(s) to use, for example,

```
ftp> put  
(local-file) localfile  
(remote-file) remotefile
```

If you omit the remote file name, the put command will create a file on the remote machine with the same name as the file on the local machine.

pwd

The **pwd** command cause *ftp* to print the name of the current working directory on the remote machine to which you are currently connected.

quit

(The same as the **bye** command above)

quote

The **quote** command causes the arguments you enter to be sent to the remote machine for execution. Arguments must be *ftp* protocol commands and arguments. The *ftp* protocol commands that a remote host supports may be displayed with the **remotehelp** command. You may enter the command string to be sent when invoking the command, for example,

```
ftp> quote NLST
```

or you may just use the command name and have *ftp* prompt you for the command line to use, for example,

```
ftp> quote  
(command line to send) NLST
```

You should not use this command unless asked to do so by your system administrator.

**recv** (The same as the **get** command above)

**remotehelp** The **remotehelp** command requests help from *ftp* at the remote machine to which you are currently connected. The information returned by the remote machine indicates which *ftp* commands it can support.

**rename** The **rename** command renames a file on the remote machine to which you are currently connected. You may enter the file names to be used when invoking the command, for example,

```
ftp> rename remotefile1 remotefile2
```

or you may just use the command name and have *ftp* prompt you for the file names to use, for example,

```
ftp> rename
(from-name) remotefile1
(to-name) remotefile2
```

**rmdir** The **rmdir** command removes a directory on the remote machine to which you are currently connected. You may specify the name of the directory to be removed when invoking the command for example,

```
ftp> rmdir /u/mydir
```

or you may just use the command name and have *ftp* prompt you for the directory name, for example,

```
ftp> rmdir
(directory-name) /u/mydir
```

Not all *ftp* servers support the **rmdir** command. *send*

**sendport** The **sendport** command causes *ftp* to disable specifying a local port to the remote machine for a data connection. The next time you enter the **sendport** command, specification of local ports will be reenabled. The default mode for local port specification when *ftp* is invoked is on. You should not use this command unless asked to do so by your system administrator.

**status** The **status** command causes *ftp* to display its current status on your terminal. This status includes the modes you select with the **bell**, **form**, **hash**, **glob**, **port**, **prompt**, and **type** commands.

- type**            The **type** command sets the file transfer type to one that you specify. Valid values are ASCII and binary. The type command is another way of invoking the ASCII and binary commands. If you do not specify a type when invoking this command, ASCII is used.
- trace**           The **trace** command causes *ftp* to enable packet tracing. The next time you enter the trace command, specification packet tracing will be disabled. You should not use this command unless asked to do so by your system administrator.
- user**            The **user** command allows you to identify yourself to the remote host when establishing a connection. If autologin was not disabled with the **-n** option when invoking *ftp*, this command is not required. If autologin is disabled or an autologin is not configured for you on the remote machine, you will have to use the **user** command to tell the remote machine who you are.

Three pieces of information are used to tell the remote machine who you are: a login name, a password, and an account name.

User name is required for all machines, password and account name are required only by some systems. Your system administrator can tell you what information is required by what machines and what are valid user and account names and passwords for a machine you wish to communicate with.

(See "Using the **.netrc** File For Automatic Login," above.)

You may enter the information to be used with the user command when invoking the command, for example,

```
ftp> user mike cat myaccount
```

Also you may just use the command name and have *ftp* prompt you for the information to use, for example,

```
ftp> user
(username) mike
password:
Account: myaccount
```

Note that *ftp* will not echo your password when you type it to protect the security of this information. If a password or account is not required on the remote machine with which you are connecting, the password and account prompts will not be displayed.

**verbose**      The **verbose** command causes *ftp* to enable verbose mode. The next time you enter the verbose command, verbose mode will be disabled. In verbose mode, the *ftp* protocol messages sent by the remote machine to *ftp* are displayed on your terminal. Also, if you use verbose mode, statistics are displayed after the completion of each file transfer. If you do not use verbose mode, this information is not displayed.

**?**              (Another name for the help command.)

### 3.6 SAMPLE FTP SESSIONS

This sample session illustrates how *ftp* can be used. Three hosts are used in these sessions, the local host **HERE** and the remote hosts **THERE**.

#### 3.6.1 Description of Session 1

This is a simple session illustrating *ftp* use for sending and receiving files. *Ftp* is invoked with a host name and automatically logs the user into that host since the **-n** (disable autologin) option was not used.

The user first turns off verbose mode. Then they change the working directory on the remote machine to the **/etc** directory. Since the **-d** (debug) option was not used, and since verbose is turned off, no messages other than the *ftp* prompt are displayed by *ftp*.

The user does a directory listing of the **/etc** directory on **THERE** using the **ls** command for an abbreviated listing. *Ftp* shows four files in **/etc** on **THERE**. The command **get "passwd"** is then issued to copy the file **passwd** from **THERE** to **HERE**. A file named **passwd** is created on **HERE** since no local file name was specified.

The **put** command is then used to copy a file called **wall** from the current working directory on the local machine to the remote working directory (**/etc**) on the remote machine (**THERE**). Once again, the same file name is used since no remote file name was specified. After the transfer is complete, a

directory listing is requested which now shows five files in **/etc** on **THERE** including the file **wall** which was just sent from **HERE**.

The **bye** command is then used to exit *ftp* and return to the shell.

```
$ ftp THERE
Connected to THERE
220 THERE FTP server (Version 4.160 #1) ready.
Name (THERE:stevea):
Password (THERE:stevea):
331 Password required for stevea.
230 User stevea logged in.
ftp> verbose off
Verbose mode off.
ftp> cd /etc
ftp> ls
passwd
volcopy
whodo
ftp> get passwd
ftp> put wall
ftp> ls
passwd
volcopy
wall
whodo
ftp> bye
$
```

### 3.6.2 Description of Session 2

This session illustrates the displays caused by using a number of *ftp* options. After invoking *ftp* with the remote host name, the user issues commands to turn on debugging. *Ftp* displays a message indicating that this option is now enabled.

The user then changes the remote working directory to **/etc**. Since debug and verbose modes are on, *ftp* displays messages showing the command sent to the remote machine, (**---> CWD /etc**), and the response received from the remote machine, (**200 CWD command okay.**). Note that the **cd** command, which has a form the same as UNIX's change directory command, is sent as a CWD command (for change working directory) to the remote machine. The CWD command is *ftp*'s way of saying **cd** independently of any specific operating system command language.

Following the **cd** command, the user does a **pwd** command to verify the working directory. Once again, *ftp* displays the messages sent between the local and remote machines and then displays the current remote working directory. The user then turns on the **hash** option. *Ftp* displays a message indicating that this option is now enabled.

The command "get wall myfile" tells *ftp* to retrieve the file wall and place it in the file myfile in the user's local working directory. *Ftp* displays the messages sent between the two hosts to begin the transfer and then prints a hash mark for each block (1024 bytes) of information received. After the transfer is complete, statistics are displayed showing the total time required and the data rate for the transfer.

After the file is received, the user closes the connection with the close command and exits *ftp* with the **bye** command.

```
$ ftp THERE
Connected to THERE
220 THERE FTP server (Version 4.160 #1) ready.
Name (THERE:stevea):
Password (THERE:stevea):
331 Password required for stevea.
ftp> debug
Debugging on (debug = 1)
ftp> cd /etc
---> CWD /etc
200 CWD command okay.
ftp> pwd
---> PWD
251 "/etc" is the current directory.
ftp> hash
Hash mark printing on (1024 bytes/hash mark).
ftp> get wall myfile
---> PORT 3,20,0,2,4,51
200 PORT command okay.
---> RETR wall
150 Opening data connection for wall (3.20.0.2,1075) (24384 bytes).
#####
226 Transfer complete.
24550 bytes received in 12.00 seconds (2 Kbytes/s)
ftp> close
---> QUIT
221 Goodbye.
ftp> bye
$
```

## 4. UNIX FILE COPY - THE RCP COMMAND

The *rcp* command allows you to copy files between any two UNIX machines on the internet. *Rcp* is similar to *ftp* but has a syntax much like the UNIX *cp* command. This command may only be used with remote machines running UNIX or a compatible operating system.

### 4.1 INVOKING RCP

Rcp is invoked from the UNIX shell. You must specify the name of local files to copy and where they are to be copied to, for example,

```
# rcp admin:/etc/hosts hosts.admin
```

As shown, file names for *rcp* follow a convention that is an extension of the UNIX file name convention. File names may take one of three forms, where a file name names a file or a directory. Valid forms for file names are:

- user@machine:filename
- machine:filename
- filename

where,

machine	is the name of the machine which contains or will contain the file. If you do not specify a machine, the file is assumed to reside on your local machine.
user	is the user name on the machine you specify. If you do not specify a user name, your user name on your local machine is used. Whether you use your user name or another user name, you must have established permission for yourself on the machine where the file is located. The system administrator of the remote machine can advise you on how the remote machine is configured.
filename	is a standard UNIX file name which may include a directory path. If the filename you specify does not begin with a slash (/), the filename is assumed to be relative to the specified user's home directory. The filename may include wild cards but these may have to be quoted to prevent their expansion on your local machine.

An exclamation point may be used in place of the colon in *rcp* filenames.

If you specify only a directory name for the destination of an *rcp* command, the file(s) you specify are copied into that directory with the same names as the files copied.

## 4.2 OPTIONS TO RCP

You can specify an option when invoking *rcp*.

**-r** This option allows the copying of directory trees. If the file specified for copying is a directory and you specify **-r**, the entire directory tree under that directory is copied. When **-r** is specified, the destination of the *rcp* command must be a directory. When you do not specify the **-r** option, requesting the copying of a directory is an error.

## 4.3 SAMPLE RCP SESSIONS

In the following examples, two remote machines on the network are used named THERE-C and THERE-C1.

The first example copies a file named **list** from the user's current directory to her home directory on THERE-C:

```
# rcp list THERE-C:list
```

The next example copies a directory hierarchy to a directory tree rooted in **src** within user's home directory on THERE-C.

```
# rcp -r /net/src THERE-C:src
```

This example shows the user copying a file from the home directory of a user named **mike** on THERE-C to the **/usr/tmp** directory on THERE-C1. The copy on THERE-C1 is to belong to a user named **deb**.

```
# rcp mike@THERE-C:list deb@THERE-C1:/usr/tmp
```

## **Chapter 5**

## **GLOSSARY**

1000000

1000000

## TCP/IP GLOSSARY

alias	An alias is an alternate host name, which can be created as a convenience in addressing a host on a local network whose unique primary name is long and/or complicated.
ARP	Address Resolution Protocol is used by Ethernet for address mapping.
ARPA	Now called DARPA, stands for Defense Advanced Research Projects Agency. ARPANET is the network based on the work sponsored by this agency. See also <b>DDN</b>
block	A block (noun) is usually 1024 bytes.
broadcast network	A broadcast network is one that "broadcasts" all transmissions instead of from point to point. Each node then "grabs" the transmissions intended for them. For example, Ethernet broadcasts down its bus.
BSD	Berkeley Software Distribution
bus	A set of parallel signals implemented in hardware in a standard manner so that multiple devices can access it and communicate over it.
connection	A connection is a logical communication path identified by a pair of sockets.
DARPA	Department of Defense Advanced Research Project Agency, formerly called ARPA. This agency sponsored the network architecture research project upon which ARPANET is based. ARPANET is a large governmental internetwork, called the Internet, part of which is the Defense Data Network (DDN ). See also <b>DDN</b> and <b>Internet</b> .
data link level	Data link level is the communications protocol for the physical media-link used to transport the data.
datagram	A datagram is a message sent in a packet switched computer communications network. The message made up of source and destination addresses and the data itself. The datagram model implies that no connection, such as a virtual circuit is needed, to

send them and that they are not required to be delivered reliably or in sequence. See also **packet**.

DDN

Defense Data Network. The Defense Data Network (DDN) is a set of communications capabilities which links together computer systems within the Department of Defense (DoD). The DDN allows users of these computer systems to send mail and files between systems and to access other computers on the network in interactive terminal sessions. The DDN is part of the DARPA Internet. See **Internet**

daemon

A daemon is a UNIX system service. It is a program that is active in the background but not connected to a terminal.

destination

The destination address, an internet header field.

destination address

The destination address, usually the network and host identifiers.

flags

Flags is an internet header field carrying various control flags.

flow control

Flow control is the function and process of regulating the traffic and amount of data between flowing nodes so that neither node is sent more data than it can handle at a given time

fragment

A fragment is an IP packet that is one part of a UDP or TCP message. Messages may be split up, or fragmented, into fragments by IP if the message length exceeds the capability of the data link.

gateway

A software service installed at a switching node that connects two or more networks, especially if they use different protocols. A gateway provides UNIX internetworking with an extended logical network by transparently attaching one or more physical networks.

header

A header is the control information at the beginning of a message, segment, datagram, fragment, packet or block of data.

host

A host is a computer. In particular a source or destination of messages from the point of view of the communication network.

ICMP	Internet Control Message Protocol. ICMP is used by a gateway or destination host to communicate with a source host, for example, to report an error in datagram processing. ICMP, uses the basic support of IP as if ICMP were a higher level protocol, however, ICMP is actually an integral part of IP, and must be implemented by every IP module.
Identification	Identification is an Internet Protocol field. This identifying value assigned by the sender aids in assembling the fragments of a datagram.
install	Install means to move the executable files from the distribution media to the system disk. In context, install can also mean to perform all the steps necessary to make a server or protocol operative
Internet	The Internet (spelled with initial capitalization) is the DARPA Internet System. See <b>DARPA</b>
Internet Address	Address is a source or destination address specific to the host level. It consists of a four octet (32 bit) source or destination address consisting of a Network field and a Local Address field.
Internet datagram	An internet datagram is the unit of data exchanged between a pair of internet modules (includes the internet header).
internet module	An internet module is an instance or individual implementation of the Internet Protocol, residing at a local host or gateway.
Internet Protocol	Internet Protocol (IP) is the network level protocol used by UNIX internetworking
internetwork	An internetwork is a supernetwork made up of two or more networks able to communicate with each other through gateways. See <b>gateway</b>
IP	See <b>Internet Protocol</b>
layer	A layer is a network function or set of related network functions that forms an autonomous functional block in the superset of network architectural functions. This method of partitioning the necessary network functions allows each layer to interface transparently to adjoining layers and thereby provides a method of making network

components more manageable.

Local Address	The Local Address the address of a host within a network. The actual mapping of an internet local address on to the host addresses in a network is quite general, allowing for many to one mappings.
local packet	A local packet is the unit of transmission within a local network.
machine	A machine is a host computer. The use of this term is similar to "host," and "node," but "machine" connotes the machine-specific or hardware aspects of the host computer, whereas "node" connotes the logical aspects of a network host. Host connotes the relationship of the local node machine to application systems and remote hosts.
module	A module is an implementation, usually in software, of a protocol or other procedure
network	A network is a collection of computer nodes able to communicate with each other
network interface	A network interface is the hardware and driver software that connects a host to a physical network.
octet	An octet is an eight bit byte.
OSI	(Open Systems Interconnection). OSI is a standard of the ISO. This standard attempts to provide for consistent hardware and software interfaces among network products. OSI and other standard setters such as IBM and the National Bureau of Standards generally divide network architecture into seven layers: physical, link, network, transport, session, presentation, and application.
packet	A packet is a package of data with a header which may or may not be logically complete. More often a physical packaging than a logical packaging of data.
point-to-point	Point-to-point is a network configuration in which two points are connected to each other by a dedicated line, which can be a direct cable connection, a leased line, or a dialup to a service providing dedicated lines
port	A port is the portion of a socket that specifies which logical input or output channel of a process is

associated with the data.

process	A process is a program in execution. A source or destination of data from the point of view of the TCP or other host-to-host protocol.
protocol	A protocol, in general, is a set of rules that enable a network entity to understand a communicating entity; however, the entity that employs these rules, such as the transport level protocol, TCP, is commonly referred to as a protocol. Therefore, a protocol is a software entity that implements a specific layer or function in a network architecture. In using the programmatic interface, a protocol is the next higher level protocol identifier, an internet header field.
RFC	Request For Comment. These refer to the "official" specification of Internet protocols and addressing mechanisms which are published by the Network Information Center of SRI in Palo Alto, CA and take the form of requests for comment.
root	Root is the login name of the super user. The super user is the user who has the widest form of machine privileges.
routing	Dynamic, or adaptive, routing is the ability to transfer data automatically to the destination node via alternative paths consisting of one or intermediate nodes. Routing includes the ability to ascertain available paths and to decide the best path, taking into account topology changes or node failures as they occur
server	A server is a system service, called a demon. It is usually a user program that runs in background, in user space, to provide a defined set of functions to the user who uses it through the command interface. Each time a user invokes it, the server provides a separate process for that user
socket	A socket is a file descriptor made up of system of data structures and pointers used by the kernel to identify and keep track of a process. It is an address which specifically includes a port identifier, that is, the concatenation of an Internet Address with a TCP port. Sockets are transparent to the user.

Programs must open sockets to access network functions. Any one process cannot have more than 20 open files at a given time.

superuser

See **root**

TCP

Transmission Control Protocol is a transport level, connection-oriented protocol that provides reliable end-to-end message transmission over an internetwork

tuple

A tuple is a mathematical term for set of numbers composed of two or more factors. For example: [(XY)(AB)].

UDP

User datagram protocol is an unreliable user level transport protocol for transaction-oriented applications. It handles datagram sockets. It uses the IP for network services.

user

The user of the internet protocol. This may be a higher level protocol module, an application program, or a gateway program.

X.25

X.25 is a circuit-switched network protocol used commonly in Europe and less so in the United States. X.25 is based on a three-layer, peer-communications protocol standard defined by the International Telegraph and Telephone Consultative Committee (CCITT).

## **Chapter 6**

### **USER'S REFERENCE**



**NAME**

intro — introduction to networking commands

**DESCRIPTION**

This section describes publicly accessible networking utilities in alphabetical order.

**SEE ALSO**

Section (1M) for network administration commands.

**DIAGNOSTICS**

Upon termination each command returns two bytes of status, one supplied by the system giving the cause for termination, and (in the case of 'normal' termination) one supplied by the program, see *wait* and *exit(2)*. The former byte is 0 for normal termination, the latter is customarily 0 for successful execution, nonzero to indicate troubles such as erroneous parameters, bad or inaccessible data, or other inability to cope with the task at hand. It is called variously 'exit code', 'exit status' or 'return code', and is described only where special conventions are involved.

**NAME**

*finger* — user information lookup program

**SYNOPSIS**

**finger** [ options ] name ...

**DESCRIPTION**

By default *finger* lists the login name, full name, terminal name and write status (as a '\*' before the terminal name if write permission is denied), idle time, login time, and office location and phone number (if they are known) for each current UNIX user. (Idle time is minutes if it is a single integer, hours and minutes if a ':' is present, or days and hours if a 'd' is present.)

A longer format also exists and is used by *finger* whenever a list of people's names is given. (Account names as well as first and last names of users are accepted.) This format is multi-line, and includes all the information described above as well as the user's home directory and login shell, any plan which the person has placed in the file *.plan* in their home directory, and the project on which they are working from the file *.project* also in the home directory.

*Finger* may be used to lookup users on a remote machine. The format is to specify the user as "user@host." If the user name is left off, the standard format listing is provided on the remote machine.

*Finger* options include:

- m Match arguments only on user name.
- l Force long output format.
- p Suppress printing of the *.plan* files
- s Force short output format.

**FILES**

<i>/etc/utmp</i>	who file (current users)
<i>/etc/wtmp</i>	who file (past logins)
<i>/etc/passwd</i>	for users names, offices, ...
<i>\$HOME/.lastlogin</i>	last login information
<i>\$HOME/.plan</i>	plans
<i>\$HOME/.project</i>	projects

**SEE ALSO**

who(1), fingerd(1M).

**BUGS**

Only the first line of the *.project* file is printed.

There is no way to pass arguments to the remote machine as *finger* uses an internet standard port.

**NAME**

*ftp* — ARPANET file transfer program

**SYNOPSIS**

**ftp** [ **-v** ] [ **-d** ] [ **-i** ] [ **-n** ] [ **-g** ] [ **host** ]

**DESCRIPTION**

*Ftp* is the user interface to the ARPANET standard File Transfer Protocol. The program allows a user to transfer files to and from a remote network site.

The client host with which *ftp* is to communicate may be specified on the command line. If this is done, *ftp* will immediately attempt to establish a connection to an FTP server on that host; otherwise, *ftp* will enter its command interpreter and await instructions from the user. When *ftp* is awaiting commands from the user the prompt *ftp>* is provided to the user. The following commands are recognized by *ftp*:

**!** [ *command* [ *args* ] ]

Invoke an interactive shell on the local machine. If there are arguments, the first is taken to be a command to execute directly, with the rest of the arguments as its arguments.

**\$** *macro-name* [ *args* ]

Execute the macro *macro-name* that was defined with the **macdef** command. Arguments are passed to the macro unglobbed.

**account** [ *passwd* ]

Supply a supplemental password required by a remote system for access to resources once a login has been successfully completed. If no argument is included, the user will be prompted for an account password in a non-echoing input mode.

**append** *local-file* [ *remote-file* ]

Append a local file to a file on the remote machine. If *remote-file* is left unspecified, the local file name is used in naming the remote file after being altered by any *ntrans* or *nmap* setting. File transfer uses the current settings for *type*, *format*, *mode*, and *structure*.

**ascii** Set the file transfer *type* to network ASCII. This is the default type.

**bell** Arrange that a bell be sounded after each file transfer command is completed.

**binary**

Set the file transfer *type* to support binary image transfer.

**bye** Terminate the FTP session with the remote server and exit *ftp*. An end of file will also terminate the session and exit.

**case** Toggle remote computer file name case mapping during **mget** commands. When **case** is on (default is off), remote computer file names with all letters in upper case are written in the local directory with the letters mapped to lower case.

**cd** *remote-directory*

Change the working directory on the remote machine to *remote-directory*.

- cdup** Change the remote machine working directory to the parent of the current remote machine working directory.
- close** Terminate the FTP session with the remote server, and return to the command interpreter. Any defined macros are erased.
- cr** Toggle carriage return stripping during ascii type file retrieval. Records are denoted by a carriage return/linefeed sequence during ascii type file transfer. When **cr** is on (the default), carriage returns are stripped from this sequence to conform with the UNIX single linefeed record delimiter. Records on non-UNIX remote systems may contain single linefeeds; when an ascii type transfer is made, these linefeeds may be distinguished from a record delimiter only when **cr** is off.
- delete** *remote-file*  
Delete the file *remote-file* on the remote machine.
- debug** [*debug-value*]  
Toggle debugging mode. If an optional *debug-value* is specified it is used to set the debugging level. When debugging is on, *ftp* prints each command sent to the remote machine, preceded by the string -->.
- dir** [*remote-directory*] [*local-file*]  
Print a listing of the directory contents in the directory, *remote-directory*, and, optionally, placing the output in *local-file*. If no directory is specified, the current working directory on the remote machine is used. If no local file is specified, or *local-file* is -, output comes to the terminal.
- disconnect**  
A synonym for **close**.
- form** *format*  
Set the file transfer *form* to *format*. The default format is file.
- get** *remote-file* [*local-file*]  
Retrieve the *remote-file* and store it on the local machine. If the local file name is not specified, it is given the same name it has on the remote machine, subject to alteration by the current *case*, *ntrans*, and *nmap* settings. The current settings for *type*, *form*, *mode*, and *structure* are used while transferring the file.
- glob** Toggle filename expansion for **mdelete**, **mget** and **mput**. If globbing is turned off with **glob**, the file name arguments are taken literally and not expanded. Globbing for **mput** is done as in **sh**(1). For **mdelete** and **mget**, each remote file name is expanded separately on the remote machine and the lists are not merged. Expansion of a directory name is likely to be different from expansion of the name of an ordinary file: the exact result depends on the foreign operating system and ftp server, and can be previewed by doing '**mls remote-files -**'. Note: **mget** and **mput** are not meant to transfer entire directory subtrees of files. That can be done by transferring a **tar**(1) archive of the subtree (in binary mode).

**hash** Toggle hash-sign ("#") printing for each data block transferred. The size of a data block is BUFSIZ bytes. BUFSIZ is defined in *<stdio.h>*.

**help** [ *command* ]

Print an informative message about the meaning of *command*. If no argument is given, *ftp* prints a list of the known commands.

**lcd** [ *directory* ]

Change the working directory on the local machine. If no *directory* is specified, the user's home directory is used.

**ls** [ *remote-directory* ] [ *local-file* ]

Print an abbreviated listing of the contents of a directory on the remote machine. If *remote-directory* is left unspecified, the current working directory is used. If no local file is specified, or if *local-file* is -, the output is sent to the terminal.

**macrodef** *macro-name*

Define a macro. Subsequent lines are stored as the macro *macro-name*; a null line (consecutive newline characters in a file or carriage returns from the terminal) terminates macro input mode. There is a limit of 16 macros and 4096 total characters in all defined macros. Macros remain defined until a **close** command is executed. The macro processor interprets '\$' and '\' as special characters. A '\$' followed by a number (or numbers) is replaced by the corresponding argument on the macro invocation command line. A '\$' followed by an 'i' signals that macro processor that the executing macro is to be looped. On the first pass '\$i' is replaced by the first argument on the macro invocation command line, on the second pass it is replaced by the second argument, and so on. A '\' followed by any character is replaced by that character. Use the '\' to prevent special treatment of the '\$'.

**mdelete** [ *remote-files* ]

Delete the *remote-files* on the remote machine.

**mdir** *remote-files local-file*

Like **dir**, except multiple remote files may be specified. If interactive prompting is on, *ftp* will prompt the user to verify that the last argument is indeed the target local file for receiving **mdir** output.

**mget** *remote-files*

Expand the *remote-files* on the remote machine and do a **get** for each file name thus produced. See **glob** for details on the filename expansion. Resulting file names will then be processed according to *case*, *ntrans*, and *nmap* settings. Files are transferred into the local working directory, which can be changed with '**lcd** *directory*'; new local directories can be created with '**! mkdir** *directory*'.

**mkdir** *directory-name*

Make a directory on the remote machine.

**mls** *remote-files local-file*

Like **ls**, except multiple remote files may be specified. If interactive prompting is on, *ftp* will prompt the user to verify that the last argument is indeed the target local file for receiving **mls** output.

**mode** [ *mode-name* ]

Set the file transfer *mode* to *mode-name*. The default mode is stream mode.

**mput** *local-files*

Expand wild cards in the list of local files given as arguments and do a **put** for each file in the resulting list. See **glob** for details of filename expansion. Resulting file names will then be processed according to *ntrans* and *nmap* settings.

**nmap** [ *inpattern outpattern* ]

Set or unset the filename mapping mechanism. If no arguments are specified, the filename mapping mechanism is unset. If arguments are specified, remote filenames are mapped during **mput** commands and **put** commands issued without a specified remote target filename. If arguments are specified, local filenames are mapped during **mget** commands and **get** commands issued without a specified local target filename. This command is useful when connecting to a non-UNIX remote computer with different file naming conventions or practices. The mapping follows the pattern set by *inpattern* and *outpattern*. *Inpattern* is a template for incoming filenames (which may have already been processed according to the *ntrans* and *case* settings). Variable templating is accomplished by including the sequences '\$1', '\$2', ..., '\$9' in *inpattern*. Use '\ ' to prevent this special treatment of the '\$' character. All other characters are treated literally, and are used to determine the *nmap inpattern* variable values. For example, given *inpattern* \$1.\$2 and the remote file name "mydata.data", \$1 would have the value "mydata", and \$2 would have the value "data". The *outpattern* determines the resulting mapped filename. The sequences '\$1', '\$2', ..., '\$9' are replaced by any value resulting from the *inpattern* template. The sequence '\$0' is replaced by the original filename. Additionally, the sequence '[seq1,seq2]' is replaced by *seq1* if *seq1* is not a null string; otherwise it is replaced by *seq2*. For example, the command "nmap \$1.\$2.\$3 [\$1,\$2].[\$2,file]" would yield the output filename "myfile.data" for input filenames "myfile.data" and "myfile.data.old", "myfile.file" for the input filename "myfile", and "myfile.myfile" for the input filename ".myfile". Spaces may be included in *outpattern*, as in the example: nmap \$1 | sed "s/ \*\$//>" > \$1 . Use the '\ ' character to prevent special treatment of the '\$', '[', ']', and ',' characters.

**ntrans** [ *inchars* [ *outchars* ] ]

Set or unset the filename character translation mechanism. If no arguments are specified, the filename character translation mechanism is unset. If arguments are specified, characters in remote filenames are translated during **mput** commands and **put** commands issued without a specified remote target filename. If arguments are specified, characters in local filenames are translated during **mget** commands and **get** commands issued without a specified local target filename. This command is useful when connecting to a non-UNIX remote computer with different file naming conventions or practices. Characters in a filename matching a character in *inchars* are replaced with the corresponding character in *outchars*. If the

character's position in *inchars* is longer than the length of *outchars*, the character is deleted from the file name.

**open** *host* [ *port* ]

Establish a connection to the specified *host* FTP server. An optional port number may be supplied, in which case, *ftp* will attempt to contact an FTP server at that port. If the *auto-login* option is on (default), *ftp* will also attempt to automatically log the user in to the FTP server (see below).

**prompt**

Toggle interactive prompting. Interactive prompting occurs during multiple file transfers to allow the user to selectively retrieve or store files. If prompting is turned off (default is on), any **mget** or **mput** will transfer all files, and any **mdelete** will delete all files.

**proxy** *ftp-command*

Execute an ftp command on a secondary control connection. This command allows simultaneous connection to two remote ftp servers for transferring files between the two servers. The first **proxy** command should be an **open**, to establish the secondary control connection. Enter the command "proxy ?" to see other ftp commands executable on the secondary connection. The following commands behave differently when prefaced by **proxy**: **open** will not define new macros during the auto-login process, **close** will not erase existing macro definitions, **get** and **mget** transfer files from the host on the primary control connection to the host on the secondary control connection, and **put**, **mput**, and **append** transfer files from the host on the secondary control connection to the host on the primary control connection. Third party file transfers depend upon support of the ftp protocol PASV command by the server on the secondary control connection.

**put** *local-file* [ *remote-file* ]

Store a local file on the remote machine. If *remote-file* is left unspecified, the local file name is used after processing according to any *ntrans* or *nmap* settings in naming the remote file. File transfer uses the current settings for *type*, *format*, *mode*, and *structure*.

**pwd** Print the name of the current working directory on the remote machine.

**quit** A synonym for **bye**.

**quote** *arg1 arg2 ...*

The arguments specified are sent, verbatim, to the remote FTP server.

**recv** *remote-file* [ *local-file* ]

A synonym for **get**.

**remotehelp** [ *command-name* ]

Request help from the remote FTP server. If a *command-name* is specified it is supplied to the server as well.

**rename** [ *from* ] [ *to* ]

Rename the file *from* on the remote machine, to the file *to*.

- reset** Clear reply queue. This command re-synchronizes command/reply sequencing with the remote ftp server. Resynchronization may be necessary following a violation of the ftp protocol by the remote server.
- rmdir** *directory-name*  
Delete a directory on the remote machine.
- runique**  
Toggle storing of files on the local system with unique filenames. If a file already exists with a name equal to the target local filename for a **get** or **mget** command, a ".1" is appended to the name. If the resulting name matches another existing file, a ".2" is appended to the original name. If this process continues up to ".99", an error message is printed, and the transfer does not take place. The generated unique filename will be reported. Note that **runique** will not affect local files generated from a shell command (see below). The default value is off.
- send** *local-file* [ *remote-file* ]  
A synonym for **put**.
- sendport**  
Toggle the use of PORT commands. By default, *ftp* will attempt to use a PORT command when establishing a connection for each data transfer. The use of PORT commands can prevent delays when performing multiple file transfers. If the PORT command fails, *ftp* will use the default data port. When the use of PORT commands is disabled, no attempt will be made to use PORT commands for each data transfer. This is useful for certain FTP implementations which do ignore PORT commands but, incorrectly, indicate they've been accepted.
- status** Show the current status of *ftp*.
- struct** [ *struct-name* ]  
Set the file transfer *structure* to *struct-name*. By default stream structure is used.
- sunique**  
Toggle storing of files on remote machine under unique file names. Remote ftp server must support ftp protocol STOU command for successful completion. The remote server will report unique name. Default value is off.
- tenex** Set the file transfer type to that needed to talk to TENEX machines.
- trace** Toggle packet tracing.
- type** [ *type-name* ]  
Set the file transfer *type* to *type-name*. If no type is specified, the current type is printed. The default type is network ASCII.
- user** *user-name* [ *password* ] [ *account* ]  
Identify yourself to the remote FTP server. If the password is not specified and the server requires it, *ftp* will prompt the user for it (after disabling local echo). If an account field is not specified, and the FTP server requires it, the user will be prompted for it. If an

account field is specified, an account command will be relayed to the remote server after the login sequence is completed if the remote server did not require it for logging in. Unless *ftp* is invoked with auto-login disabled, this process is done automatically on initial connection to the FTP server.

**verbose**

Toggle verbose mode. In verbose mode, all responses from the FTP server are displayed to the user. In addition, if verbose is on, when a file transfer completes, statistics regarding the efficiency of the transfer are reported. By default, verbose is on.

**xmkdir** *directory-name*

Make a directory on the remote machine. This sends an XMKD command instead of MKD, and is useful for backwards compatibility with 4.2BSD UNIX machines.

**xpwd** Print the name of the current working directory on the remote machine. This sends an XPWD command instead of PWD, and is useful for backwards compatibility with 4.2BSD UNIX machines.**xrmdir** *directory-name*

Delete a directory on the remote machine. This sends an XRMD command instead of RMD, and is useful for backwards compatibility with 4.2BSD UNIX machines.

**? [ command ]**

A synonym for help.

Command arguments which have embedded spaces may be quoted with quote (") marks.

**ABORTING A FILE TRANSFER**

To abort a file transfer, use the terminal interrupt key (usually Ctrl-C). Sending transfers will be immediately halted. Receiving transfers will be halted by sending a ftp protocol ABOR command to the remote server, and discarding any further data received. The speed at which this is accomplished depends upon the remote server's support for ABOR processing. If the remote server does not support the ABOR command, an "ftp>" prompt will not appear until the remote server has completed sending the requested file.

The terminal interrupt key sequence will be ignored when *ftp* has completed any local processing and is awaiting a reply from the remote server. A long delay in this mode may result from the ABOR processing described above, or from unexpected behavior by the remote server, including violations of the ftp protocol. If the delay results from unexpected remote server behavior, the local *ftp* program must be killed by hand.

**FILE NAMING CONVENTIONS**

Files specified as arguments to *ftp* commands are processed according to the following rules.

- 1) If the file name — is specified, the **stdin** (for reading) or **stdout** (for writing) is used.
- 2) If the first character of the file name is |, the remainder of the argument is interpreted as a shell command. *Ftp* then forks a shell,

using *popen*(3) with the argument supplied, and reads (writes) from the stdout (stdin). If the shell command includes spaces, the argument must be quoted; e.g. "*| ls -lt*". A particularly useful example of this mechanism is: *dir |more*.

- 3) Failing the above checks, if "globbing" is enabled, local file names are expanded according to the rules used in the *sh*(1); c.f. the *glob* command. If the *ftp* command expects a single local file (e.g. **put**), only the first filename generated by the "globbing" operation is used.
- 4) For **mget** commands and **get** commands with unspecified local file names, the local filename is the remote filename, which may be altered by a **case**, **ntrans**, or **nmap** setting. The resulting filename may then be altered if **runique** is on.
- 5) For **mput** commands and **put** commands with unspecified remote file names, the remote filename is the local filename, which may be altered by a **ntrans** or **nmap** setting. The resulting filename may then be altered by the remote server if **sunique** is on.

#### FILE TRANSFER PARAMETERS

The FTP specification specifies many parameters which may affect a file transfer. The *type* may be one of *ascii*, *image* (binary), *ebcdic*, and *local byte size* (for PDP-10's and PDP-20's mostly). *Ftp* supports the *ascii* and *image* types of file transfer, plus *local byte size 8* for **tenex** mode transfers.

*Ftp* supports only the default values for the remaining file transfer parameters: *mode*, *form*, and *struct*.

#### OPTIONS

Options may be specified at the command line, or to the command interpreter.

The **-v** (verbose on) option forces *ftp* to show all responses from the remote server, as well as report on data transfer statistics. Normally, this is on by default, unless the standard input is not a terminal.

The **-n** option restrains *ftp* from attempting auto-login upon initial connection. If auto-login is enabled, *ftp* will check the *.netrc* (see below) file in the user's home directory for an entry describing an account on the remote machine. If no entry exists, *ftp* will prompt for the remote machine login name (default is the user identity on the local machine), and, if necessary, prompt for a password and an account with which to login.

The **-i** option turns off interactive prompting during multiple file transfers.

The **-d** option enables debugging.

The **-g** option disables file name globbing.

#### THE *.netrc* FILE

The *.netrc* file contains login and initialization information used by the auto-login process. It resides in the user's home directory. The following tokens are recognized; they may be separated by spaces, tabs, or new-lines:

##### **machine name**

Identify a remote machine name. The auto-login process searches the *.netrc* file for a **machine** token that matches the remote machine specified on the *ftp* command line or as an **open** command

argument. Once a match is made, the subsequent `.netrc` tokens are processed, stopping when the end of file is reached or another **machine** token is encountered.

**login** *name*

Identify a user on the remote machine. If this token is present, the auto-login process will initiate a login using the specified name.

**password** *string*

Supply a password. If this token is present, the auto-login process will supply the specified string if the remote server requires a password as part of the login process. Note that if this token is present in the `.netrc` file, *ftp* will abort the auto-login process if the `.netrc` is readable by anyone besides the user.

**account** *string*

Supply an additional account password. If this token is present, the auto-login process will supply the specified string if the remote server requires an additional account password, or the auto-login process will initiate an ACCT command if it does not.

**macdef** *name*

Define a macro. This token functions like the *ftp* **macdef** command functions. A macro is defined with the specified name; its contents begin with the next `.netrc` line and continue until a null line (consecutive new-line characters) is encountered. If a macro named *init* is defined, it is automatically executed as the last step in the auto-login process.

**BUGS**

Correct execution of many commands depends upon proper behavior by the remote server.

An error in the treatment of carriage returns in the 4.2BSD UNIX `ascii-mode` transfer code has been corrected. This correction may result in incorrect transfers of binary files to and from 4.2BSD servers using the `ascii` type. Avoid this problem by using the `binary` image type.

**HOSTNAME(1)**

**(TCP/IP)**

**HOSTNAME(1)**

**NAME**

hostname — set or print name of current host system

**SYNOPSIS**

**hostname** [ nameofhost ]

**DESCRIPTION**

The *hostname* command prints the name of the current host, as given before the “login” prompt. The super-user can set the hostname by giving an argument; this is usually done at boot time in a startup script.

**SEE ALSO**

gethostname(3), sethostname(3), uname(1).

**NAME**

logger - make entries in the system log

**SYNOPSIS**

logger [ -t tag ] [ -p pri ] [ -i ] [ -f file ] [ message ... ]

**ARGUMENTS**

-t *tag*      Mark every line in the log with the specified *tag*.  
-p *pri*      Enter the message with the specified priority. The priority may be specified numerically or as a "facility.level" pair. For example, "-p local3.info" logs the message(s) as *informational* level in the *local3* facility. The default is "user.notice."  
-i            Log the process id of the logger process with each line.  
-f *file*      Log the specified file.  
message      The message to log; if not specified, the -f file or standard input is logged.

**DESCRIPTION**

*Logger* provides a program interface to the *syslog(3)* system log module.

A message can be given on the command line, which is logged immediately, or a file is read and each line is logged.

**EXAMPLES**

logger System rebooted

logger -p local0.notice -t OPER -f /tmp/msg

**SEE ALSO**

syslog(3), syslogd(1M).

## NAME

netstat — show network status

## SYNOPSIS

```
netstat [ -AaimnrS ] [ -f address_family ] [ -I interface ] [ -p
protocol_name ] [ interval ] [ namelist ] [ corefile ]
```

## DESCRIPTION

The *netstat* command symbolically displays the contents of various network-related data structures. The options have the following meanings:

- A show the address of any associated protocol control blocks; used for debugging
- a show the state of all sockets; normally sockets used by server processes are not shown
- i show the state of interfaces which have been auto-configured (interfaces statically configured into a system, but not located at boot time are not shown)
- m show network memory usage
- n show network addresses as numbers (normally *netstat* interprets addresses and attempts to display them symbolically)
- s show per-protocol statistics
- r show the routing tables
- S show serial line configuration
- f limit statistics and control block displays to *address-family*. The only address-family currently supported is **inet**
- I show interface state for *interface* only.
- p limit statistics and control block displays to *protocol-name*, e.g. **tcp**.

The arguments *namelist* and *corefile* allow substitutes for the defaults **/unix** and **/dev/kmem**.

If an *interval* is specified, *netstat* will continuously display the information regarding packet traffic on the configured network interfaces, pausing *interval* seconds before refreshing the screen.

There are a number of display formats, depending on the information presented. The default display, for active sockets, shows the local and remote addresses, send and receive queue sizes (in bytes), protocol, and, optionally, the internal state of the protocol.

Address formats are of the form “host.port” or “network.port” if a socket’s address specifies a network but no specific host address. When known, the host and network addresses are displayed symbolically according to the data bases **/etc/hosts** and **/etc/networks**, respectively. If a symbolic name for an address is unknown, or if the **—n** option is specified, the address is printed in the Internet “dot format”; refer to *rhosts*(4) for more information regarding this format. Unspecified, or “wildcard,” addresses and ports appear as “\*”

The interface display provides a table of cumulative statistics regarding packets transferred, errors, and collisions. The network address (currently

Internet specific) of the interface and the maximum transmission unit ("mtu") are also displayed.

The routing table display indicates the available routes and their status. Each route consists of a destination host or network and a gateway to use in forwarding packets. The "flags" field shows the state of the route ("U" if "up"), and whether the route is to a gateway ("G"). Direct routes are created for each interface attached to the local host. The "refcnt" field gives the current number of active uses of the route. Connection-oriented protocols normally hold on to a single route for the duration of a connection, while connectionless protocols obtain a route then discard it. The use field provides a count of the number of packets sent using that route. The interface entry indicates the network interface utilized for the route.

When *netstat* is invoked with an *interval* argument, it displays a running count of statistics related to network interfaces. This display consists of a column summarizing information for all interfaces and a column for the interface with the most traffic since the system was last rebooted. The first line of each screen of information contains a summary since the system was last rebooted. Subsequent lines of output show values accumulated over the preceding interval.

The serial line display shows the mapping of serial line units to serial devices. The baud rate and protocols in use are also shown.

**SEE ALSO**

slattach(1M), hosts(4), networks(4), protocols(4), services(4).

**BUGS**

Interface statistics are dependent on the link driver. If it does not attach itself to the *ifstats* structure in the kernel, the message "No Statistics Available" will be printed for that interface.

**NAME**

nslookup — query name servers interactively

**SYNOPSIS**

**nslookup** [ *host-to-find* ] — [ *server address* | *server name* ]]

**DESCRIPTION**

*Nslookup* is a program to query DARPA Internet domain name servers. Nslookup has two modes: interactive and non-interactive. Interactive mode allows the user to query the name server for information about various hosts and domains or print a list of hosts in the domain. Non-interactive mode is used to print just the name and Internet address of a host or domain.

**ARGUMENTS**

Interactive mode is entered in the following cases:

- a) when no arguments are given (the default name server will be used), and
- b) when the first argument is a hyphen (—) and the second argument is the host name of a name server.

Non-interactive mode is used when the name of the host to be looked up is given as the first argument. The optional second argument specifies a name server.

**INTERACTIVE COMMANDS**

Commands may be interrupted at any time by typing a control-C. To exit, type a control-D (EOF). The command line length must be less than 80 characters. **N.B.** an unrecognized command will be interpreted as a host name.

host [server]

Look up information for *host* using the current default server or using *server* if it is specified.

**server** *domain*

**lserver** *domain*

Change the default server to *domain*. **Lserver** uses the initial server to look up information about *domain* while **server** uses the current default server. If an authoritative answer can't be found, the names of servers that might have the answer are returned.

**root** Changes the default server to the server for the root of the domain name space. Currently, the host sri-nic.arpa is used. (This command is a synonym for the **lserver sri-nic.arpa**.) The name of the root server can be changed with the **set root** command.

**finger** [*name*] [> *filename*]

**finger** [*name*] [>> *filename*]

Connects with the finger server on the current host. The current host is defined when a previous lookup for a host was successful and

returned address information (see the **set querytype=A** command). *Name* is optional. > and >> can be used to redirect output in the usual manner.

```
ls domain > filename]
ls domain >> filename]
ls -a domain > filename]
ls -a domain >> filename]
ls -h domain > filename]
ls -h domain >> filename]
ls -d domain > filename]
```

List the information available for *domain*. The default output contains host names and their Internet addresses. The **-a** option lists aliases of hosts in the domain. The **-h** option lists CPU and operating system information for the domain. The **-d** option lists all contents of a zone transfer. When output is directed to a file, hash marks are printed for every 50 records received from the server.

#### view *filename*

Sorts and lists the output of previous **ls** command(s) with *more*(1).

#### help

? Prints a brief summary of commands.

#### set *keyword*[=*value*]

This command is used to change state information that affects the lookups. Valid keywords are:

**all** Prints the current values of the various options to **set**. Information about the current default server and host is also printed.

#### [no]debug

Turn debugging mode on. A lot more information is printed about the packet sent to the server and the resulting answer.

(Default = nodebug, abbreviation = [no]deb)

**[no]d2** Turn exhaustive debugging mode on. Essentially all fields of every packet are printed.

(Default = nod2)

#### [no]defname

Append the default domain name to every lookup.

(Default = defname, abbreviation = [no]def)

#### [no]search

With defname, search for each name in parent domains of the current domain.

(Default = search)

#### domain=*name*

Change the default domain name to *name*. The default

domain name is appended to all lookup requests if the **def-name** option has been set. The search list is set to parents of the domain with at least two components in their names. (Default = value in `hostname` or `/etc/resolv.conf`, abbreviation = `do`)

**querytype=***value*

**type=***value*

Change the type of information returned from a query to one of:

<b>A</b>	the host's Internet address (the default).
<b>CNAME</b>	the canonical name for an alias.
<b>HINFO</b>	the host CPU and operating system type.
<b>MD</b>	the mail destination.
<b>MX</b>	the mail exchanger.
<b>MG</b>	the mail group member.
<b>MINFO</b>	the mailbox or mail list information.
<b>MR</b>	the mail rename domain name.
<b>NS</b>	nameserver for the named zone.

Other types specified in the RFC1035 document are valid but aren't very useful.

(Abbreviation = `q`)

**[no]recurse**

Tell the name server to query other servers if it does not have the information.

(Default = `recurse`, abbreviation = `[no]rec`)

**retry=***number*

Set the number of retries to *number*. When a reply to a request is not received within a certain amount of time (changed with **set timeout**), the request is resent. The retry value controls how many times a request is resent before giving up.

(Default = 2, abbreviation = `ret`)

**root=***host*

Change the name of the root server to *host*. This affects the **root** command.

(Default = `sri-nic.arpa`, abbreviation = `ro`)

**timeout=***number*

Change the time-out interval for waiting for a reply to *number* seconds.

(Default = 10 seconds, abbreviation = `t`)

**[no]vc** Always use a virtual circuit when sending requests to the server.

(Default = `novc`, abbreviation = `[no]v`)

## DIAGNOSTICS

If the lookup request was not successful, an error message is printed.

Possible errors are:

**Time-out**

The server did not respond to a request after a certain amount of time (changed with **set timeout=value**) and a certain number of retries (changed with **set retry=value**).

**No information**

Depending on the query type set with the **set querytype** command, no information about the host was available, though the host name is valid.

**Non-existent domain**

The host or domain name does not exist.

**Connection refused**

**Network is unreachable**

The connection to the name or finger server could not be made at the current time. This error commonly occurs with **finger** requests.

**Server failure**

The name server found an internal inconsistency in its database and could not return a valid answer.

**Refused**

The name server refused to service the request.

The following error should not occur and it indicates a bug in the program.

**Format error**

The name server found that the request packet was not in the proper format.

**FILES**

/etc/resolv.conf            initial domain name and name server addresses.  
/usr/local/lib/nslookup.hlp    Help file

**SEE ALSO**

resolver(3), resolver(4), named(1M), RFC974, RFC1034, RFC1035.

**NAME**

`rcmd` — remote shell command execution

**SYNOPSIS**

```
rcmd node [-l user] [-n] [command]
/usr/hosts/node [-l user] [-n] [command]
```

**DESCRIPTION**

`rcmd` sends *command* to *node* for execution. It passes the resulting remote command its own standard input and outputs the remote command's standard output and standard error. *Command* can consist of more than one parameter. The second, simplified form of the command is equivalent to the first, but is only available if the system administrator previously ran *mkhosts*(1M). Interrupt, quit, and terminate signals received by *rcmd* are also received by the remote command; *rcmd* normally terminates at the same time as the remote command.

If *command* is omitted, *rcmd* simply runs *rlogin*(1).

By default, the command belongs to the user on the remote node with the same name as the user who ran *rcmd*. This means that the resulting processes belong to the remote user and begin with the remote user's home directory as their working directory. Options permit you to specify another user on *node* as the owner. In any case, the remote system must have declared the local user equivalent to the remote user: an entry in */etc/hosts.equiv* or in a *.rhosts* file in the current directory (normally the home directory) of the target user will demonstrate equivalence. [See *rcmd*(3).]

*rcmd* understands the following options:

- `-l user` The command is to belong to *user* on *node*.
  - `-n` Prevent the remote command from blocking on input by making its standard input be */dev/null* instead of *rcmd*'s standard input.
- If `-n` is not specified, *rcmd* reads the local standard input regardless of whether the remote machine reads standard input.

**EXAMPLES**

The following command runs *who* on a node called "central," putting the output in a file on the local machine.

```
rcmd central who > /tmp/c.who
```

The next example puts the same output on the remote machine.

```
rcmd central who \> /tmp/c.who
```

**FILES**

*\$HOME/.rhosts* (on the target machine)  
*/etc/hosts.equiv* (on the target machine)

**SEE ALSO**

*mkhosts*(1M), *rlogin*(1), *rshd*(1M), *rhosts*(4).

**REQUIREMENTS**

*rshd*(1M) must be running on the target machine.

**NOTE**

In some installations, this command is called **rsh**, so as to be like other versions of the software.

**WARNINGS**

As the above examples illustrate, metacharacters to be interpreted by the remote shell must be hidden from the local shell. Thus

```
rcmd central cd /etc ; cat passwd
```

clearly doesn't do what was intended because the semicolon is interpreted by the local shell, not the remote shell, and the remote shell never even sees the **cat** command. Either of the following commands properly escapes the semicolon:

```
rcmd central cd /etc \; cat passwd
rcmd central 'cd /etc ; cat passwd'
```

**NAME**

*rcp* — remote file copy

**SYNOPSIS**

*rcp* [ *-r* ] [ *-p* ] *file1* [ *file2* ... ] *target*

**DESCRIPTION**

*rcp* copies files between two nodes. *rcp* works like the *cp* command (see *cp*(1)), with some extensions.

*File1* is copied to *target*. If *target* is a directory, one or more files are copied into that directory; the copies have the same names as the originals.

File and directory names follow a convention which is an extension of the normal UNIX convention. Names take one of four forms:

```
@ .ds |F @ .}S 2 3 "user@" "host" ":" "path" "" ""
node.user:path
host:path
path
```

where

*host* is the name of the system which contains or will contain the file. If no *host* is specified (the simple *path* form of the name), the system on which the command is executed is assumed.

*user* is the name of a user on the specified system. If no user is specified (the *host:path* and *path* forms of the name), the user on the remote system whose name is the same as the user who executed the *rcp* command is used.

Access to the file system is as if by the specified user who has just logged in. Created files belong to the specified user and the specified user's group (taken from the password file). File and directory modifications can only occur if the specified user has permission to do them. If *path* does not begin with a slant (/), it is assumed to be relative to the specified user's home directory.

To use a user name on a remote system, the remote system must have declared it "equivalent" to your user name. See *rhsts*(4).

*path* is a conventional UNIX path name. *Path* can include file name generation sequences (\*, ?, [...]); it may be necessary to quote these to prevent their expansion on the local system.

The *-r* (recursive) option copies directory hierarchies. If a file specified for copying is a directory and *-r* is specified, the entire hierarchy under it is copied. When *-r* is specified, *target* must be a directory.

When *-r* is not specified, copying directories is an error.

By default, the mode and owner of *file2* are preserved if it already existed; otherwise the mode of the source file modified by the *umask*(2) on the destination host is used. The *-p* option causes *rcp* to attempt to preserve (duplicate) in its copies the modification times and modes of the source files, ignoring the *umask*.

Note that a third system (not the source or target system of the copy) can execute *rcp*.

### EXAMPLES

The following examples are executed on system alpha, by user fred. Alpha is networked to beta and gamma.

The first example copies *list* from fred's home directory on alpha to fred's home directory on beta.

```
rcp list beta:list
```

The next example copies a directory hierarchy. The original is rooted at *src* in fred's home directory on beta. The copy is to be rooted in *src* in the working directory.

```
rcp -r beta:src .
```

Finally, fred copies a file from diane's home directory on beta to */usr/tmp* on gamma; the copy on gamma is to belong to karl. Both diane and karl must have previously declared fred on alpha equivalent to their own user names; see *rhsts*(4).

```
rcp beta.diane:junk gamma.karl:/usr/tmp
```

Note that *junk* is not placed in karl's home directory because the *path* part of the name begins with a slash.

### FILES

```
/etc/hosts.equiv  
$HOME/.rhosts
```

### SEE ALSO

*ftp*(1).

### REQUIREMENTS

Both nodes involved in the copy must be running the *rshd*(1M) server.

### DIAGNOSTICS

Most diagnostics are self-explanatory. "Permission denied" means either that the remote user does not have permission to do what you want or that the remote user is not equivalent to you.

### WARNINGS

If a remote shell invoked by *rcp* has output on startup, *rcp* will get confused. This is never a problem with *sh*(1), because it is not called as a login shell.

The *-r* option doesn't work correctly if the copy is purely local, since it relies on underlying support from *cp*, which is only available on BSD derived systems. Use *cpio*(1), instead.

**NAME**

rlogin — remote login

**SYNOPSIS**

```
rlogin rhost [ -e c ] [ -8 ] [ -L ] [ -l username ]
/usr/hosts/rhost [ -e c ] [ -8 ] [ -L ] [ -l username ]
```

**DESCRIPTION**

*Rlogin* connects your terminal on the current local host system *lhost* to the remote host system *rhost*.

Each host has a file */etc/hosts.equiv* which contains a list of *rhost*'s with which it shares account names. (The host names must be the standard names as described in *rcmd*(1).) When you *rlogin* as the same user on an equivalent host, you don't need to give a password. Each user may also have a private equivalence list in a file *.rhosts* in his login directory. Each line in this file should contain an *rhost* and a *username* separated by a space, giving additional cases where logins without passwords are to be permitted. If the originating user is not equivalent to the remote user, then a login and password will be prompted for on the remote machine as in *login*(1). To avoid some security problems, the *.rhosts* file must be owned by either the remote user or root.

The remote terminal type is the same as your local terminal type (as given in your environment *TERM* variable). The terminal or window size is also copied to the remote system if the server supports the option, and changes in size are reflected as well. All echoing takes place at the remote site, so that (except for delays) the rlogin is transparent. Flow control via *^S* and *^Q* and flushing of input and output on interrupts are handled properly. The optional argument **-8** allows an eight-bit input data path at all times; otherwise parity bits are stripped except when the remote side's stop and start characters are other than *^S/^Q*. The argument **-L** allows the rlogin session to be run in without any output post-processing, (e.g. *stty -opost*.) A line of the form "*~.*" disconnects from the remote host, where "*~*" is the escape character. A different escape character may be specified by the **-e** option. There is no space separating this option flag and the argument character.

**SEE ALSO**

netlogin(1M), rcmd(1), rlogind(1M), rhosts(4).

**FILES**

*/usr/hosts/\** for *rhost* version of the command

**BUGS**

More of the environment should be propagated.

**RUPTIME(1)**

**(TCP/IP)**

**RUPTIME(1)**

**NAME**

ruptime — show host status of local machines

**SYNOPSIS**

**ruptime** [ **-a** ] [ **-r** ] [ **-l** ] [ **-t** ] [ **-u** ]

**DESCRIPTION**

*Ruptime* gives a status line for each machine on the local network; these are formed from packets broadcast by each host on the network once every 1 - 3 minutes.

Machines for which no status report has been received for 5 minutes are shown as being down.

Users idle an hour or more are not counted unless the **-a** flag is given.

Normally, the listing is sorted by host name. The **-l**, **-t**, and **-u** flags specify sorting by load average, uptime, and number of users, respectively. The **-r** flag reverses the sort order.

**FILES**

/usr/spool/rwho/whod.\*      data files

**SEE ALSO**

rwho(1), rwhod(1M).

**NAME**

*rwho* - who is logged in on local network

**SYNOPSIS**

*rwho* [ -a ]

**DESCRIPTION**

*Rwho* lists users logged in on machines on the local network. The format is similar to that of *who*(1). Without options, only users who have typed in the last hour are listed. For each user listed, *rwho* displays the user name; the host name; and the date and time the user logged in. If the user has not typed in the last minute, *rwho* also displays the user's idle time in hours and minutes.

*Rwho* understands the following option:

- a List all users on active nodes (users idle for more than an hour are listed).

If information from a host is more than five minutes old, the host is assumed to be down and its users are not listed.

**REQUIREMENTS**

Each host to be listed must be running the *rwod*(1M) server, which broadcasts a status packet once every 1 - 3 minutes. The local host must also be running this server to maintain the data files. Since broadcasts do not cross gateways, hosts on other networks will not be listed.

**FILES**

/usr/spool/rwho/whod.\* information about other nodes

**SEE ALSO**

*ruptime*(1), *rwod*(1M).

**NAME**

talk — talk to another user

**SYNOPSIS**

**talk** person [ ttyname ]

**DESCRIPTION**

*Talk* is a visual communication program which copies lines from your terminal to that of another user.

If you wish to talk to someone on you own machine, then *person* is just the person's login name. If you wish to talk to a user on another host, then *person* is of the form :

*host!user* or  
*host.user* or  
*host:user* or  
 @ *user@host*

though @ *user@host* is perhaps preferred.

If you want to talk to a user who is logged in more than once, the *ttyname* argument may be used to indicate the appropriate terminal name.

When first called, it sends the message

```
Message from TalkDaemon@his_machine...
talk: connection requested by your_name@your_machine.
talk: respond with: talk your_name@your_machine
```

to the user you wish to talk to. At this point, the recipient of the message should reply by typing

```
talk your_name@your_machine
```

It doesn't matter from which machine the recipient replies, as long as his login-name is the same. Once communication is established, the two parties may type simultaneously, with their output appearing in separate windows. Typing control L will cause the screen to be reprinted, while your erase and kill characters will work in talk as normal. In addition, control-W is defined as a word-kill character. To exit, just type your interrupt character; *talk* then moves the cursor to the bottom of the screen and restores the terminal.

Permission to talk may be denied or granted by use of the *mesg(1)* command. At the outset talking is allowed. Certain commands, in particular *nroff(1)* and *pr(1)* disallow messages in order to prevent messy output.

**FILES**

/etc/hosts	to find the recipient's machine
/etc/utmp	to find the recipient's tty

**SEE ALSO**

*mesg(1)*, *who(1)*, *mail(1)*, *write(1)*, *talkd(1M)*.

**BUGS**

The version of *talk(1)* released with System V STREAMS TCP uses a protocol that is incompatible with the protocol used in the version released with 4.2BSD. The new protocol is compatible with 4.3BSD. The older protocol was not portable across different machine architectures.

**TALK(1)**

**(TCP/IP)**

**TALK(1)**

Talk may be confused if you attempt to use the host.user format with a fully qualified hostname.

**NAME**

telnet — user interface to the TELNET protocol

**SYNOPSIS**

telnet [ host [ port ] ]

**DESCRIPTION**

*Telnet* is used to communicate with another host using the **TELNET** protocol. If *telnet* is invoked without arguments, it enters command mode, indicated by its prompt (*telnet>*). In this mode, it accepts and executes the commands listed below. If it is invoked with arguments, it performs an **open** command (see below) with those arguments.

Once a connection has been opened, *telnet* enters an input mode. The input mode entered will be either character at a time or line by line depending on what the remote system supports.

In character at a time mode, most text typed is immediately sent to the remote host for processing.

In line by line mode, all text is echoed locally, and (normally) only completed lines are sent to the remote host. The local echo character (initially **^E**) may be used to turn off and on the local echo (this would mostly be used to enter passwords without the password being echoed).

In either mode, if the *localchars* toggle is **TRUE** (the default in line mode; see below), the user's *quit*, *intr*, and *flush* characters are trapped locally, and sent as **TELNET** protocol sequences to the remote side. There are options (see *toggle autoflush* and *toggle autosynch* below) which cause this action to flush subsequent output to the terminal (until the remote host acknowledges the **TELNET** sequence) and flush previous terminal input (in the case of *quit* and *intr*).

While connected to a remote host, *telnet* command mode may be entered by typing the *telnet* escape character (initially **^]**). When in command mode, the normal terminal editing conventions are available.

**COMMANDS**

The following commands are available. Only enough of each command to uniquely identify it need be typed (this is also true for arguments to the **mode**, **set**, **toggle**, and **display** commands).

**open** *host* [ *port* ]

Open a connection to the named host. If no port number is specified, *telnet* will attempt to contact a **TELNET** server at the default port. The host specification may be either a host name (see *hosts(5)*) or an Internet address specified in the dot notation (see *inet(3N)*).

**close**

Close a **TELNET** session and return to command mode.

**quit**

Close any open **TELNET** session and exit *telnet*. An end of file (in command mode) will also close a session and exit.

**z**

Suspend *telnet*. On System V systems, this command provides the

user with an escape to a shell running on the local machine.

**mode type**

*Type* is either *line* (for line by line mode) or *character* (for character at a time mode). The remote host is asked for permission to go into the requested mode. If the remote host is capable of entering that mode, the requested mode will be entered.

**status**

Show the current status of *telnet*. This includes the peer one is connected to, as well as the current mode. In addition, both the local and remote **TELNET** options in effect are shown.

**display [ argument... ]**

Displays all, or some, of the **set** and **toggle** values (see below).

**? [ command ]**

Get help. With no arguments, *telnet* prints a help summary. If a command is specified, *telnet* will print the help information for just that command.

**send arguments**

Sends one or more special character sequences to the remote host. The following are the arguments which may be specified (more than one argument may be specified at a time):

*escape*

Sends the current *telnet* escape character (initially ^).

*synch*

Sends the **TELNET SYNCH** sequence. This sequence causes the remote system to discard all previously typed (but not yet read) input. This sequence is sent as TCP urgent data (and may not work if the remote system is a 4.2 BSD system -- if it doesn't work, a lower case r may be echoed on the terminal).

*brk*

Sends the **TELNET BRK** (Break) sequence, which may have significance to the remote system.

*ip*

Sends the **TELNET IP** (Interrupt Process) sequence, which should cause the remote system to abort the currently running process.

*ao*

Sends the **TELNET AO** (Abort Output) sequence, which should cause the remote system to flush all output **from** the remote system **to** the user's terminal.

*ayt*

Sends the **TELNET AYT** (Are You There) sequence, to which the remote system may or may not choose to respond.

*ec*

Sends the **TELNET EC** (Erase Character) sequence, which should cause the remote system to erase the last character entered.

*el*

Sends the **TELNET EL** (Erase Line) sequence, which should cause the remote system to erase the line currently being entered.

*ga*

Sends the **TELNET GA** (Go Ahead) sequence, which likely has no significance to the remote system.

*nop*

Sends the **TELNET NOP** (No OPeration) sequence.

*?*

Prints out help information for the **send** command.

**set argument value**

Set any one of a number of *telnet* variables to a specific value. The special value **off** turns off the function associated with the variable. The values of variables may be interrogated with the **display** command. The variables which may be specified are:

*echo*

This is the value (initially **^E**) which, when in line by line mode, toggles between doing local echoing of entered characters (for normal processing), and suppressing echoing of entered characters (for entering, say, a password).

*escape*

This is the *telnet* escape character (initially **^]**) which causes entry into *telnet* command mode (when connected to a remote system).

*interrupt*

If *telnet* is in *localchars* mode (see **toggle localchars** below) and the *interrupt* character is typed, a **TELNET IP** sequence (see **send ip** above) is sent to the remote host. The initial value for the interrupt character is taken to be the terminal's **intr** character.

*quit*

If *telnet* is in *localchars* mode (see **toggle localchars** below) and the *quit* character is typed, a **TELNET BRK** sequence (see **send brk** above) is sent to the remote host. The initial value for the quit character is taken to be the terminal's **quit** character.

*flushoutput*

If *telnet* is in *localchars* mode (see **toggle localchars** below) and the *flushoutput* character is typed, a **TELNET AO** sequence (see **send ao** above) is sent to the remote host. The initial value for the flush character is taken to be the terminal's **flush** character.

*erase*

If *telnet* is in *localchars* mode (see **toggle localchars** below), and if *telnet* is operating in character at a time mode, then when this character is typed, a **TELNET EC** sequence (see

**send** *ec* above) is sent to the remote system. The initial value for the erase character is taken to be the terminal's **erase** character.

**kill**

If *telnet* is in *localchars* mode (see **toggle** *localchars* below), and if *telnet* is operating in character at a time mode, then when this character is typed, a **TELNET EL** sequence (see **send** *el* above) is sent to the remote system. The initial value for the kill character is taken to be the terminal's **kill** character.

**eof**

If *telnet* is operating in line by line mode, entering this character as the first character on a line will cause this character to be sent to the remote system. The initial value of the eof character is taken to be the terminal's **eof** character.

**toggle arguments...**

Toggle (between TRUE and FALSE) various flags that control how *telnet* responds to events. More than one argument may be specified. The state of these flags may be interrogated with the **display** command. Valid arguments are:

**localchars**

If this is TRUE, then the *flush*, *interrupt*, *quit*, *erase*, and *kill* characters (see **set** above) are recognized locally, and transformed into (hopefully) appropriate **TELNET** control sequences (respectively *ao*, *ip*, *brk*, *ec*, and *el*; see **send** above). The initial value for this toggle is TRUE in line by line mode, and FALSE in character at a time mode.

**autoflush**

If *autoflush* and *localchars* are both TRUE, then when the *ao*, *intr*, or *quit* characters are recognized (and transformed into **TELNET** sequences; see **set** above for details), *telnet* refuses to display any data on the user's terminal until the remote system acknowledges (via a **TELNET Timing Mark** option) that it has processed those **TELNET** sequences. The initial value for this toggle is TRUE if the terminal user had not done an "stty noflush", otherwise FALSE (see *stty(1)*).

**autosynch**

If *autosynch* and *localchars* are both TRUE, then when either the *intr* or *quit* characters is typed (see **set** above for descriptions of the *intr* and *quit* characters), the resulting **TELNET** sequence sent is followed by the **TELNET SYNCH** sequence. This procedure **should** cause the remote system to begin throwing away all previously typed input until both of the **TELNET** sequences have been read and acted upon. The initial value of this toggle is FALSE.

**crmod**

Toggle carriage return mode. When this mode is enabled, most carriage return characters received from the remote host will be mapped into a carriage return followed by a line

feed. This mode does not affect those characters typed by the user, only those received from the remote host. This mode is not very useful unless the remote host only sends carriage return, but never line feed. The initial value for this toggle is FALSE.

*debug*

Toggles socket level debugging (useful only to the *superuser*). The initial value for this toggle is FALSE.

*options*

Toggles the display of some internal *telnet* protocol processing (having to do with **TELNET** options). The initial value for this toggle is FALSE.

*netdata*

Toggles the display of all network data (in hexadecimal format). The initial value for this toggle is FALSE.

*?*

Displays the legal **toggle** commands.

**do option****dont option****will option****wont option**

These commands allow the user to send the appropriate **TELNET** option sequence. If no option is specified, *telnet* will prompt for one.

**BUGS**

There is no adequate way for dealing with flow control.

On some remote systems, echo has to be turned off manually when in line by line mode.

There is enough settable state to justify a *.telnetrc* file.

No capability for a *.telnetrc* file is provided.

In line by line mode, the terminal's *eof* character is only recognized (and sent to the remote system) when it is the first character on a line.

**NAME**

*tftp* — user interface to the DARPA TFTP protocol

**SYNOPSIS**

*tftp* [ *host* [ *port* ] ]

**DESCRIPTION**

*tftp* is the user interface to the DARPA standard Trivial File Transfer Protocol. The program allows a user to transfer files to and from a remote network site.

The client host with which *tftp* is to communicate may be specified on the command line. If this is done, *tftp* will immediately attempt to establish a connection to a TFTP server on that host. Otherwise, *tftp* will enter its command interpreter and await instructions from the user. When *tftp* is awaiting commands from the user, the prompt

*tftp*>

appears. The following commands are recognized by *tftp*:

**connect** *host-name* [ *port* ]

Set the *host* (and optionally *port*) for transfers. Note that the TFTP protocol, unlike the FTP protocol, does not maintain connections between transfers; thus, the *connect* command does not actually create a connection, but merely remembers what host is to be used for transfers. You do not have to use the *connect* command; the remote host can be specified as part of the *get* or *put* commands.

**mode** *transfer-mode*

Set the mode for transfers; *transfer-mode* may be one of *ascii* or *binary*. The default is *ascii*.

**put** *file*

**put** *localfile remotefile*

**put** *file1 file2 ... fileN remote-directory*

Put a file or set of files to the specified remote file or directory. The destination can be in one of two forms: a filename on the remote host, if the host has already been specified, or a string of the form *host:filename* to specify both a host and filename at the same time. If the latter form is used, the hostname specified becomes the default for future transfers. If the remote-directory form is used, the remote host is assumed to be a *UNIX* machine.

**get** *filename*

**get** *remoteName localname*

**get** *file1 file2 ... fileN*

Get a file or set of files from the specified *sources*. *Source* can be in one of two forms: a filename on the remote host, if the host has already been specified, or a string of the form *host:filename* to specify both a host and filename at the same time. If the latter form is used, the last hostname specified becomes the default for future transfers.

**quit** Exit *tftp*. An end of file also exits.

**verbose**

Toggle verbose mode.

**trace** Toggle packet tracing.  
**status** Show current status.  
**rexmt** *retransmission-timeout*  
Set the per-packet retransmission timeout, in seconds.  
**timeout** *total-transmission-timeout*  
Set the total transmission timeout, in seconds.  
**ascii** Shorthand for "mode ascii"  
**binary** Shorthand for "mode binary"  
**? [ *command-name* ... ]**  
Print help information.

**FILES**

/etc/hosts

**SEE ALSO**

tftpd(1M).

**WARNINGS**

Because there is no user-login or validation within the *TFTP* protocol, the remote site will probably have some sort of file-access restrictions in place. The exact methods are specific to each site.





## REQUEST FOR READER'S COMMENTS

Intel's Technical Publications Departments attempt to provide publications that meet the needs of all Intel product users. This form lets you participate directly in the publication process. Your comments will help us correct and improve our publications. Please take a few minutes to respond.

Please restrict your comments to the usability, accuracy, organization, and completeness of this publication. If you have any comments on the product that this publication describes, please contact your Intel representative. If you wish to order publications, contact the Literature Department (see page ii of this manual).

1. Please describe any errors you found in this publication (include page number).

---

---

---

---

2. Does the publication cover the information you expected or required? Please make suggestions for improvement.

---

---

---

---

3. Is this the right type of publication for your needs? Is it at the right level? What other types of publications are needed?

---

---

---

---

4. Did you have any difficulty understanding descriptions or wording? Where?

---

---

---

---

5. Please rate this publication on a scale of 1 to 5 (5 being the best rating). \_\_\_\_\_

NAME \_\_\_\_\_ DATE \_\_\_\_\_

TITLE \_\_\_\_\_

COMPANY NAME/DEPARTMENT \_\_\_\_\_

ADDRESS \_\_\_\_\_ PHONE (     ) \_\_\_\_\_

CITY \_\_\_\_\_ STATE \_\_\_\_\_ ZIP CODE \_\_\_\_\_

(COUNTRY)

Please check here if you require a written reply. ☐



NO POSTAGE  
NECESSARY  
IF MAILED  
IN THE  
UNITED STATES



**BUSINESS REPLY MAIL**

FIRST CLASS MAIL PERMIT NO. 79 HILLSBORO, OR

POSTAGE WILL BE PAID BY ADDRESSEE

**INTEL CORPORATION**  
**IMSO TECHNICAL PUBLICATIONS** MS HF3-60  
**5200 NE ELAM YOUNG PARKWAY**  
**HILLSBORO OR 97124-9987**



Please fold here and close the card with tape. Do not staple

## INTERNATIONAL SALES OFFICES

INTEL CORPORATION  
3065 Bowers Avenue  
Santa Clara, California 95051

BELGIUM  
Intel Corporation SA  
Rue des Cottages 65  
B-1180 Brussels

DENMARK  
Intel Denmark A/S  
Glentevej 61-3rd Floor  
dk-2400 Copenhagen

ENGLAND  
Intel Corporation (U.K.) LTD.  
Piper's Way  
Swindon, Wiltshire SN3 1RJ

FINLAND  
Intel Finland OY  
Ruosilante 2  
00390 Helsinki

FRANCE  
Intel Paris  
1 Rue Edison-BP 303  
78054 St.-Quentin-en-Yvelines Cedex

ISRAEL  
Intel Semiconductor LTD.  
Atidim Industrial Park  
Neve Sharet  
P.O. Box 43202  
Tel-Aviv 61430

ITALY  
Intel Corporation S.P.A.  
Milandfiori, Palazzo E/4  
20090 Assago (Milano)

JAPAN  
Intel Japan K.K.  
Flower-Hill Shin-machi  
1-23-9, Shinmachi  
Setagaya-ku, Tokyo 15

NETHERLANDS  
Intel Semiconductor (Nederland B.V.)  
Alexanderpoort Building  
Marten Meesweg 93  
3068 Rotterdam

NORWAY  
Intel Norway A/S  
P.O. Box 92  
Hvamveien 4  
N-2013, Skjetten

SPAIN  
Intel Iberia  
Calle Zurbaran 28-IZQDA  
28010 Madrid

SWEDEN  
Intel Sweden A.B.  
Dalvaegen 24  
S-171 36 SOLNA

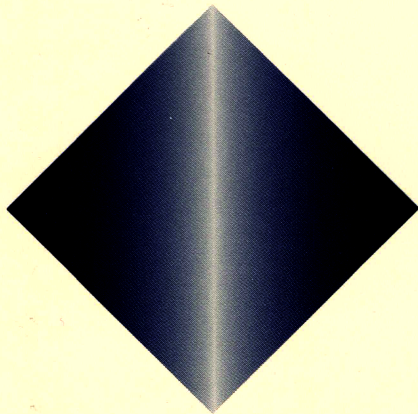
SWITZERLAND  
Intel Semiconductor A.G.  
Talackerstrasse 17  
8125 Glattbrugg  
CH-8065 Zurich

WEST GERMANY  
Intel Semiconductor G.M.B.H  
Seidlestrasse 27  
D-800 Munchen



# UNIX<sup>®</sup>

SYSTEM V RELEASE 3.2



■ Intel UNIX<sup>®</sup> System V is the result of over seven years of operating system development with AT&T. The Intel UNIX software product family includes the base operating system, extensive development tools, networking capabilities and graphical user interfaces. Intel's commitment to quality products, support, and training make Intel UNIX System V the UNIX operating system of choice for today's application developers.

■ Intel's UNIX product line continues a long-standing tradition of products that adhere to established industry standards. Moreover, Intel UNIX products are tuned for Intel's 386<sup>™</sup> and 486<sup>™</sup> microprocessor architectures. Developers and users alike can feel confident that their investment in Intel UNIX products will give them an edge in today's competitive world.

UNIX is a registered trademark of AT&T.

---

**Intel Corporation**

465727-001

3065 Bowers Avenue, Santa Clara, California 95051

